

1

Natural Language Processing

2

Jacob Eisenstein

3

June 11, 2018

4 Contents

5	Contents	1
6	1 Introduction	13
7	1.1 Natural language processing and its neighbors	13
8	1.2 Three themes in natural language processing	17
9	1.2.1 Learning and knowledge	17
10	1.2.2 Search and learning	19
11	1.2.3 Relational, compositional, and distributional perspectives	20
12	1.3 Learning to do natural language processing	22
13	1.3.1 Background	22
14	1.3.2 How to use this book	23
15	I Learning	27
16	2 Linear text classification	29
17	2.1 Naïve Bayes	32
18	2.1.1 Types and tokens	34
19	2.1.2 Prediction	35
20	2.1.3 Estimation	36
21	2.1.4 Smoothing and MAP estimation	38
22	2.1.5 Setting hyperparameters	38
23	2.2 Discriminative learning	39
24	2.2.1 Perceptron	40
25	2.2.2 Averaged perceptron	42
26	2.3 Loss functions and large-margin classification	43
27	2.3.1 Large margin classification	46
28	2.3.2 Support vector machines	47
29	2.3.3 Slack variables	48
30	2.4 Logistic regression	50
31	2.4.1 Regularization	51

32	2.4.2 Gradients	52
33	2.5 Optimization	52
34	2.5.1 Batch optimization	53
35	2.5.2 Online optimization	54
36	2.6 *Additional topics in classification	56
37	2.6.1 Feature selection by regularization	56
38	2.6.2 Other views of logistic regression	56
39	2.7 Summary of learning algorithms	58
40	3 Nonlinear classification	61
41	3.1 Feedforward neural networks	62
42	3.2 Designing neural networks	64
43	3.2.1 Activation functions	64
44	3.2.2 Network structure	65
45	3.2.3 Outputs and loss functions	66
46	3.2.4 Inputs and lookup layers	67
47	3.3 Learning neural networks	67
48	3.3.1 Backpropagation	69
49	3.3.2 Regularization and dropout	71
50	3.3.3 *Learning theory	72
51	3.3.4 Tricks	73
52	3.4 Convolutional neural networks	75
53	4 Linguistic applications of classification	81
54	4.1 Sentiment and opinion analysis	81
55	4.1.1 Related problems	83
56	4.1.2 Alternative approaches to sentiment analysis	84
57	4.2 Word sense disambiguation	85
58	4.2.1 How many word senses?	86
59	4.2.2 Word sense disambiguation as classification	87
60	4.3 Design decisions for text classification	88
61	4.3.1 What is a word?	88
62	4.3.2 How many words?	91
63	4.3.3 Count or binary?	92
64	4.4 Evaluating classifiers	92
65	4.4.1 Precision, recall, and F -MEASURE	93
66	4.4.2 Threshold-free metrics	95
67	4.4.3 Classifier comparison and statistical significance	96
68	4.4.4 *Multiple comparisons	99
69	4.5 Building datasets	99
70	4.5.1 Metadata as labels	100

71	4.5.2 Labeling data	100
72	5 Learning without supervision	107
73	5.1 Unsupervised learning	107
74	5.1.1 K -means clustering	108
75	5.1.2 Expectation Maximization (EM)	110
76	5.1.3 EM as an optimization algorithm	114
77	5.1.4 How many clusters?	115
78	5.2 Applications of expectation-maximization	116
79	5.2.1 Word sense induction	116
80	5.2.2 Semi-supervised learning	117
81	5.2.3 Multi-component modeling	118
82	5.3 Semi-supervised learning	119
83	5.3.1 Multi-view learning	120
84	5.3.2 Graph-based algorithms	121
85	5.4 Domain adaptation	122
86	5.4.1 Supervised domain adaptation	123
87	5.4.2 Unsupervised domain adaptation	124
88	5.5 *Other approaches to learning with latent variables	126
89	5.5.1 Sampling	126
90	5.5.2 Spectral learning	128
91	II Sequences and trees	135
92	6 Language models	137
93	6.1 N -gram language models	138
94	6.2 Smoothing and discounting	141
95	6.2.1 Smoothing	141
96	6.2.2 Discounting and backoff	142
97	6.2.3 *Interpolation	143
98	6.2.4 *Kneser-Ney smoothing	145
99	6.3 Recurrent neural network language models	146
100	6.3.1 Backpropagation through time	148
101	6.3.2 Hyperparameters	149
102	6.3.3 Gated recurrent neural networks	149
103	6.4 Evaluating language models	151
104	6.4.1 Held-out likelihood	151
105	6.4.2 Perplexity	152
106	6.5 Out-of-vocabulary words	153

107	7 Sequence labeling	155
108	7.1 Sequence labeling as classification	155
109	7.2 Sequence labeling as structure prediction	157
110	7.3 The Viterbi algorithm	159
111	7.3.1 Example	162
112	7.3.2 Higher-order features	163
113	7.4 Hidden Markov Models	163
114	7.4.1 Estimation	165
115	7.4.2 Inference	165
116	7.5 Discriminative sequence labeling with features	167
117	7.5.1 Structured perceptron	170
118	7.5.2 Structured support vector machines	170
119	7.5.3 Conditional random fields	172
120	7.6 Neural sequence labeling	177
121	7.6.1 Recurrent neural networks	177
122	7.6.2 Character-level models	179
123	7.6.3 Convolutional Neural Networks for Sequence Labeling	180
124	7.7 *Unsupervised sequence labeling	180
125	7.7.1 Linear dynamical systems	182
126	7.7.2 Alternative unsupervised learning methods	182
127	7.7.3 Semiring Notation and the Generalized Viterbi Algorithm	182
128	8 Applications of sequence labeling	185
129	8.1 Part-of-speech tagging	185
130	8.1.1 Parts-of-Speech	186
131	8.1.2 Accurate part-of-speech tagging	190
132	8.2 Morphosyntactic Attributes	192
133	8.3 Named Entity Recognition	193
134	8.4 Tokenization	195
135	8.5 Code switching	196
136	8.6 Dialogue acts	197
137	9 Formal language theory	199
138	9.1 Regular languages	200
139	9.1.1 Finite state acceptors	201
140	9.1.2 Morphology as a regular language	202
141	9.1.3 Weighted finite state acceptors	204
142	9.1.4 Finite state transducers	209
143	9.1.5 *Learning weighted finite state automata	214
144	9.2 Context-free languages	215
145	9.2.1 Context-free grammars	216

146	9.2.2 Natural language syntax as a context-free language	219
147	9.2.3 A phrase-structure grammar for English	221
148	9.2.4 Grammatical ambiguity	226
149	9.3 *Mildly context-sensitive languages	226
150	9.3.1 Context-sensitive phenomena in natural language	227
151	9.3.2 Combinatory categorial grammar	228
152	10 Context-free parsing	233
153	10.1 Deterministic bottom-up parsing	234
154	10.1.1 Recovering the parse tree	236
155	10.1.2 Non-binary productions	236
156	10.1.3 Complexity	237
157	10.2 Ambiguity	237
158	10.2.1 Parser evaluation	238
159	10.2.2 Local solutions	239
160	10.3 Weighted Context-Free Grammars	240
161	10.3.1 Parsing with weighted context-free grammars	241
162	10.3.2 Probabilistic context-free grammars	243
163	10.3.3 *Semiring weighted context-free grammars	245
164	10.4 Learning weighted context-free grammars	245
165	10.4.1 Probabilistic context-free grammars	246
166	10.4.2 Feature-based parsing	246
167	10.4.3 *Conditional random field parsing	247
168	10.4.4 Neural context-free grammars	249
169	10.5 Grammar refinement	250
170	10.5.1 Parent annotations and other tree transformations	251
171	10.5.2 Lexicalized context-free grammars	252
172	10.5.3 *Refinement grammars	256
173	10.6 Beyond context-free parsing	257
174	10.6.1 Reranking	257
175	10.6.2 Transition-based parsing	258
176	11 Dependency parsing	261
177	11.1 Dependency grammar	261
178	11.1.1 Heads and dependents	262
179	11.1.2 Labeled dependencies	263
180	11.1.3 Dependency subtrees and constituents	264
181	11.2 Graph-based dependency parsing	266
182	11.2.1 Graph-based parsing algorithms	268
183	11.2.2 Computing scores for dependency arcs	269
184	11.2.3 Learning	271

185	11.3 Transition-based dependency parsing	272
186	11.3.1 Transition systems for dependency parsing	273
187	11.3.2 Scoring functions for transition-based parsers	277
188	11.3.3 Learning to parse	278
189	11.4 Applications	281
190	III Meaning	285
191	12 Logical semantics	287
192	12.1 Meaning and denotation	288
193	12.2 Logical representations of meaning	289
194	12.2.1 Propositional logic	289
195	12.2.2 First-order logic	290
196	12.3 Semantic parsing and the lambda calculus	294
197	12.3.1 The lambda calculus	295
198	12.3.2 Quantification	297
199	12.4 Learning semantic parsers	299
200	12.4.1 Learning from derivations	300
201	12.4.2 Learning from logical forms	302
202	12.4.3 Learning from denotations	303
203	13 Predicate-argument semantics	309
204	13.1 Semantic roles	311
205	13.1.1 VerbNet	312
206	13.1.2 Proto-roles and PropBank	313
207	13.1.3 FrameNet	314
208	13.2 Semantic role labeling	316
209	13.2.1 Semantic role labeling as classification	316
210	13.2.2 Semantic role labeling as constrained optimization	319
211	13.2.3 Neural semantic role labeling	321
212	13.3 Abstract Meaning Representation	322
213	13.3.1 AMR Parsing	325
214	13.4 Applications of Predicate-Argument Semantics	326
215	14 Distributional and distributed semantics	333
216	14.1 The distributional hypothesis	333
217	14.2 Design decisions for word representations	335
218	14.2.1 Representation	335
219	14.2.2 Context	336
220	14.2.3 Estimation	337

221	14.3 Latent semantic analysis	338
222	14.4 Brown clusters	339
223	14.5 Neural word embeddings	343
224	14.5.1 Continuous bag-of-words (CBOW)	343
225	14.5.2 Skipgrams	344
226	14.5.3 Computational complexity	344
227	14.5.4 Word embeddings as matrix factorization	346
228	14.6 Evaluating word embeddings	347
229	14.6.1 Intrinsic evaluations	347
230	14.6.2 Extrinsic evaluations	348
231	14.7 Distributed representations beyond distributional statistics	349
232	14.7.1 Word-internal structure	350
233	14.7.2 Lexical semantic resources	352
234	14.8 Distributed representations of multiword units	352
235	14.8.1 Purely distributional methods	352
236	14.8.2 Distributional-compositional hybrids	353
237	14.8.3 Supervised compositional methods	354
238	14.8.4 Hybrid distributed-symbolic representations	355
239	15 Reference Resolution	359
240	15.1 Forms of referring expressions	360
241	15.1.1 Pronouns	360
242	15.1.2 Proper Nouns	365
243	15.1.3 Nominals	366
244	15.2 Algorithms for coreference resolution	366
245	15.2.1 Mention-pair models	367
246	15.2.2 Mention-ranking models	368
247	15.2.3 Transitive closure in mention-based models	369
248	15.2.4 Entity-based models	370
249	15.3 Representations for coreference resolution	375
250	15.3.1 Features	376
251	15.3.2 Distributed representations of mentions and entities	378
252	15.4 Evaluating coreference resolution	381
253	16 Discourse	385
254	16.1 Segments	385
255	16.1.1 Topic segmentation	386
256	16.1.2 Functional segmentation	387
257	16.2 Entities and reference	387
258	16.2.1 Centering theory	388
259	16.2.2 The entity grid	389

260	16.2.3 *Formal semantics beyond the sentence level	390
261	16.3 Relations	390
262	16.3.1 Shallow discourse relations	391
263	16.3.2 Hierarchical discourse relations	394
264	16.3.3 Argumentation	398
265	16.3.4 Applications of discourse relations	399
266	IV Applications	405
267	17 Information extraction	407
268	17.1 Entities	409
269	17.1.1 Entity linking by learning to rank	410
270	17.1.2 Collective entity linking	412
271	17.1.3 *Pairwise ranking loss functions	413
272	17.2 Relations	415
273	17.2.1 Pattern-based relation extraction	416
274	17.2.2 Relation extraction as a classification task	417
275	17.2.3 Knowledge base population	420
276	17.2.4 Open information extraction	424
277	17.3 Events	425
278	17.4 Hedges, denials, and hypotheticals	426
279	17.5 Question answering and machine reading	428
280	17.5.1 Formal semantics	428
281	17.5.2 Machine reading	429
282	18 Machine translation	435
283	18.1 Machine translation as a task	435
284	18.1.1 Evaluating translations	437
285	18.1.2 Data	439
286	18.2 Statistical machine translation	440
287	18.2.1 Statistical translation modeling	441
288	18.2.2 Estimation	443
289	18.2.3 Phrase-based translation	444
290	18.2.4 *Syntax-based translation	445
291	18.3 Neural machine translation	446
292	18.3.1 Neural attention	448
293	18.3.2 *Neural machine translation without recurrence	450
294	18.3.3 Out-of-vocabulary words	451
295	18.4 Decoding	453
296	18.5 Training towards the evaluation metric	455

297	19 Text generation	459
298	19.1 Data-to-text generation	459
299	19.1.1 Latent data-to-text alignment	461
300	19.1.2 Neural data-to-text generation	462
301	19.2 Text-to-text generation	466
302	19.2.1 Neural abstractive summarization	466
303	19.2.2 Sentence fusion for multi-document summarization	468
304	19.3 Dialogue	469
305	19.3.1 Finite-state and agenda-based dialogue systems	469
306	19.3.2 Markov decision processes	470
307	19.3.3 Neural chatbots	472
308	A Probability	475
309	A.1 Probabilities of event combinations	475
310	A.1.1 Probabilities of disjoint events	476
311	A.1.2 Law of total probability	477
312	A.2 Conditional probability and Bayes' rule	477
313	A.3 Independence	479
314	A.4 Random variables	480
315	A.5 Expectations	481
316	A.6 Modeling and estimation	482
317	B Numerical optimization	485
318	B.1 Gradient descent	486
319	B.2 Constrained optimization	486
320	B.3 Example: Passive-aggressive online learning	487
321	Bibliography	489

322 Notation

323 As a general rule, words, word counts, and other types of observations are indicated with
324 Roman letters (a, b, c); parameters are indicated with Greek letters (α, β, θ). Vectors are
325 indicated with bold script for both random variables \mathbf{x} and parameters $\boldsymbol{\theta}$. Other useful
326 notations are indicated in the table below.

Basics

$\exp x$	the base-2 exponent, 2^x
$\log x$	the base-2 logarithm, $\log_2 x$
$\{x_n\}_{n=1}^N$	the set $\{x_1, x_2, \dots, x_N\}$
x_i^j	x_i raised to the power j
$x_i^{(j)}$	indexing by both i and j

Linear algebra

$\mathbf{x}^{(i)}$	a column vector of feature counts for instance i , often word counts
$\mathbf{x}_{j:k}$	elements j through k (inclusive) of a vector \mathbf{x}
$[\mathbf{x}; \mathbf{y}]$	vertical concatenation of two column vectors
$[\mathbf{x}, \mathbf{y}]$	horizontal concatenation of two column vectors
\mathbf{e}_n	a “one-hot” vector with a value of 1 at position n , and zero everywhere else
$\boldsymbol{\theta}^\top$	the transpose of a column vector $\boldsymbol{\theta}$
$\boldsymbol{\theta} \cdot \mathbf{x}^{(i)}$	the dot product $\sum_{j=1}^N \theta_j \times x_j^{(i)}$
\mathbf{X}	a matrix
$x_{i,j}$	row i , column j of matrix \mathbf{X}
$\text{Diag}(\mathbf{x})$	a matrix with \mathbf{x} on the diagonal, e.g., $\begin{pmatrix} x_1 & 0 & 0 \\ 0 & x_2 & 0 \\ 0 & 0 & x_3 \end{pmatrix}$
\mathbf{X}^{-1}	the inverse of matrix \mathbf{X}

Text datasets

w_m	word token at position m
N	number of training instances
M	length of a sequence (of words or tags)
V	number of words in vocabulary
$y^{(i)}$	the true label for instance i
\hat{y}	a predicted label
\mathcal{Y}	the set of all possible labels
K	number of possible labels $K = \mathcal{Y} $
\square	the start token
\blacksquare	the stop token
$\mathbf{y}^{(i)}$	a structured label for instance i , such as a tag sequence
$\mathcal{Y}(\mathbf{w})$	the set of possible labelings for the word sequence \mathbf{w}
\diamond	the start tag
\blacklozenge	the stop tag

Probabilities

$\Pr(A)$	probability of event A
$\Pr(A B)$	probability of event A , conditioned on event B
$p_B(b)$	the marginal probability of random variable B taking value b ; written $p(b)$ when the choice of random variable is clear from context
$p_{B A}(b a)$	the probability of random variable B taking value b , conditioned on A taking value a ; written $p(b a)$ when clear from context
$A \sim p$	the random variable A is distributed according to distribution p . For example, $X \sim \mathcal{N}(0, 1)$ states that the random variable X is drawn from a normal distribution with zero mean and unit variance.
$A B \sim p$	conditioned on the random variable B , A is distributed according to p . ¹

Machine learning

$\Psi(\mathbf{x}^{(i)}, y)$	the score for assigning label y to instance i
$\mathbf{f}(\mathbf{x}^{(i)}, y)$	the feature vector for instance i with label y
θ	a (column) vector of weights
$\ell^{(i)}$	loss on an individual instance i
L	objective function for an entire dataset
\mathcal{L}	log-likelihood of a dataset
λ	the amount of regularization

327 **Chapter 1**

328 **Introduction**

329 Natural language processing is the set of methods for making human language accessible
330 to computers. In the past decade, natural language processing has become embedded
331 in our daily lives: automatic machine translation is ubiquitous on the web and in social
332 media; text classification keeps emails from collapsing under a deluge of spam; search
333 engines have moved beyond string matching and network analysis to a high degree of
334 linguistic sophistication; dialog systems provide an increasingly common and effective
335 way to get and share information.

336 These diverse applications are based on a common set of ideas, drawing on algo-
337 rithms, linguistics, logic, statistics, and more. The goal of this text is to provide a survey
338 of these foundations. The technical fun starts in the next chapter; the rest of this current
339 chapter situates natural language processing with respect to other intellectual disciplines,
340 identifies some high-level themes in contemporary natural language processing, and ad-
341 vises the reader on how best to approach the subject.

342 **1.1 Natural language processing and its neighbors**

343 One of the great pleasures of working in this field is the opportunity to draw on many
344 other intellectual traditions, from formal linguistics to statistical physics. This section
345 briefly situates natural language processing with respect to some of its closest neighbors.

346 **Computational Linguistics** Most of the meetings and journals that host natural lan-
347 guage processing research bear the name “computational linguistics”, and the terms may
348 be thought of as essentially synonymous. But while there is substantial overlap, there is
349 an important difference in focus. In linguistics, language is the object of study. Compu-
350 tational methods may be brought to bear, just as in scientific disciplines like computational
351 biology and computational astronomy, but they play only a supporting role. In contrast,

352 natural language processing is focused on the design and analysis of computational al-
 353 gorithms and representations for processing natural human language. The goal of natu-
 354 ral language processing is to provide new computational capabilities around human lan-
 355 guage: for example, extracting information from texts, translating between languages, an-
 356 swering questions, holding a conversation, taking instructions, and so on. Fundamental
 357 linguistic insights may be crucial for accomplishing these tasks, but success is ultimately
 358 measured by whether and how well the job gets done.

359 **Machine Learning** Contemporary approaches to natural language processing rely heav-
 360 ily on machine learning, which makes it possible to build complex computer programs
 361 from examples. Machine learning provides an array of general techniques for tasks like
 362 converting a sequence of discrete tokens in one vocabulary to a sequence of discrete to-
 363 kens in another vocabulary — a generalization of what normal people might call “transla-
 364 tion.” Much of today’s natural language processing research can be thought of as applied
 365 machine learning. However, natural language processing has characteristics that distin-
 366 guish it from many of machine learning’s other application domains.

- 367 • Unlike images or audio, text data is fundamentally discrete, with meaning created
 368 by combinatorial arrangements of symbolic units. This is particularly consequential
 369 for applications in which text is the output, such as translation and summarization,
 370 because it is not possible to gradually approach an optimal solution.
- 371 • Although the set of words is discrete, new words are always being created. Further-
 372 more, the distribution over words (and other linguistic elements) resembles that of a
 373 **power law** (Zipf, 1949): there will be a few words that are very frequent, and a long
 374 tail of words that are rare. A consequence is that natural language processing algo-
 375 rithms must be especially robust to observations that do not occur in the training
 376 data.
- 377 • Language is **recursive**: units such as words can combine to create phrases, which
 378 can combine by the very same principles to create larger phrases. For example, a
 379 **noun phrase** can be created by combining a smaller noun phrase with a **preposi-**
 380 **tional phrase**, as in *the whiteness of the whale*. The prepositional phrase is created by
 381 combining a preposition (in this case, *of*) with another noun phrase (*the whale*). In
 382 this way, it is possible to create arbitrarily long phrases, such as,

383 (1.1) ...huge globular pieces of the whale of the bigness of a human head.¹

384 The meaning of such a phrase must be analyzed in accord with the underlying hier-
 385 archical structure. In this case, *huge globular pieces of the whale* acts as a single noun
 386 phrase, which is conjoined with the prepositional phrase of *the bigness of a human*

¹Throughout the text, this notation will be used to introduce linguistic examples.

387 *head*. The interpretation would be different if instead, *huge globular pieces* were con-
 388 joined with the prepositional phrase *of the whale of the bigness of a human head* —
 389 implying a disappointingly small whale. Even though text appears as a sequence,
 390 machine learning methods must account for its implicit recursive structure.

391 **Artificial Intelligence** The goal of artificial intelligence is to build software and robots
 392 with the same range of abilities as humans (Russell and Norvig, 2009). Natural language
 393 processing is relevant to this goal in several ways. The capacity for language is one of the
 394 central features of human intelligence, and no artificial intelligence program could be said
 395 to be complete without the ability to communicate in words.²

396 Much of artificial intelligence research is dedicated to the development of systems
 397 that can reason from premises to a conclusion, but such algorithms are only as good as
 398 what they know (Dreyfus, 1992). Natural language processing is a potential solution to
 399 the “knowledge bottleneck”, by acquiring knowledge from natural language texts, and
 400 perhaps also from conversations; This idea goes all the way back to Turing’s 1949 pa-
 401 per *Computing Machinery and Intelligence*, which proposed the **Turing test** and helped to
 402 launch the field of artificial intelligence (Turing, 2009).

403 Conversely, reasoning is sometimes essential for basic tasks of language processing,
 404 such as determining who a pronoun refers to. **Winograd schemas** are examples in which
 405 a single word changes the likely referent of a pronoun, in a way that seems to require
 406 knowledge and reasoning to decode (Levesque et al., 2011). For example,

407 (1.2) The trophy doesn’t fit into the brown suitcase because **it** is too [small/large].

408 When the final word is *small*, then the pronoun *it* refers to the suitcase; when the final
 409 word is *large*, then *it* refers to the trophy. Solving this example requires spatial reasoning;
 410 other schemas require reasoning about actions and their effects, emotions and intentions,
 411 and social conventions.

412 The Winograd schemas demonstrate that natural language understanding cannot be
 413 achieved in isolation from knowledge and reasoning. Yet the history of artificial intelli-
 414 gence has been one of increasing specialization: with the growing volume of research in
 415 subdisciplines such as natural language processing, machine learning, and computer vi-

²This view seems to be shared by some, but not all, prominent researchers in artificial intelligence. Michael Jordan, a specialist in machine learning, has said that if he had a billion dollars to spend on any large research project, he would spend it on natural language processing (https://www.reddit.com/r/MachineLearning/comments/2fxi6v/ama_michael_i_jordan/). On the other hand, in a public discussion about the future of artificial intelligence in February 2018, computer vision researcher Yann LeCun argued that language was perhaps the “50th most important” thing to work on, and that it would be a great achievement if AI could attain the capabilities of an orangutan, which presumably do not include language (<http://www.abigailsee.com/2018/02/21/deep-learning-structure-and-innate-priors.html>).

416 sion, it is difficult for anyone to maintain expertise across the entire field. Still, recent work
417 has demonstrated interesting connections between natural language processing and other
418 areas of AI, including computer vision (e.g., Antol et al., 2015) and game playing (e.g.,
419 Branavan et al., 2009). The dominance of machine learning throughout artificial intel-
420 ligence has led to a broad consensus on representations such as graphical models and
421 knowledge graphs, and on algorithms such as backpropagation and combinatorial opti-
422 mization. Many of the algorithms and representations covered in this text are part of this
423 consensus.

424 **Computer Science** The discrete and recursive nature of natural language invites the ap-
425 plication of theoretical ideas from computer science. Linguists such as Chomsky and
426 Montague have shown how formal language theory can help to explain the syntax and
427 semantics of natural language. Theoretical models such as finite-state and pushdown au-
428 tomata are the basis for many practical natural language processing systems. Algorithms
429 for searching the combinatorial space of analyses of natural language utterances can be
430 analyzed in terms of their computational complexity, and theoretically motivated approx-
431 imations can sometimes be applied.

432 The study of computer systems is also relevant to natural language processing. Pro-
433 cessing large datasets of unlabeled text is a natural application for parallelization tech-
434 niques like MapReduce (Dean and Ghemawat, 2008; Lin and Dyer, 2010); handling high-
435 volume streaming data sources such as social media is a natural application for approx-
436 imate streaming and sketching techniques (Goyal et al., 2009). When deep neural net-
437 works are implemented in production systems, it is possible to eke out speed gains using
438 techniques such as reduced-precision arithmetic (Wu et al., 2016). Many classical natu-
439 ral language processing algorithms are not naturally suited to graphics processing unit
440 (GPU) parallelization, suggesting directions for further research at the intersection of nat-
441 ural language processing and computing hardware (Yi et al., 2011).

442 **Speech Processing** Natural language is often communicated in spoken form, and speech
443 recognition is the task of converting an audio signal to text. From one perspective, this is
444 a signal processing problem, which might be viewed as a preprocessing step before nat-
445 ural language processing can be applied. However, context plays a critical role in speech
446 recognition by human listeners: knowledge of the surrounding words influences percep-
447 tion and helps to correct for noise (Miller et al., 1951). For this reason, speech recognition
448 is often integrated with text analysis, particularly with statistical **language model**, which
449 quantify the probability of a sequence of text (see chapter 6). Beyond speech recognition,
450 the broader field of speech processing includes the study of speech-based dialogue sys-
451 tems, which are briefly discussed in chapter 19. Historically, speech processing has often
452 been pursued in electrical engineering departments, while natural language processing

453 has been the purview of computer scientists. For this reason, the extent of interaction
454 between these two disciplines is less than it might otherwise be.

455 **Others** Natural language processing plays a significant role in emerging interdisciplinary
456 fields like **computational social science** and the **digital humanities**. Text classification
457 (chapter 4), clustering (chapter 5), and information extraction (chapter 17) are particularly
458 useful tools; another is probabilistic **topic models** (Blei, 2012), which are not covered in
459 this text. **Information retrieval** (Manning et al., 2008) makes use of similar tools, and
460 conversely, techniques such as latent semantic analysis (§ 14.3) have roots in information
461 retrieval. **Text mining** is sometimes used to refer to the application of data mining tech-
462 niques, especially classification and clustering, to text. While there is no clear distinction
463 between text mining and natural language processing (nor between data mining and ma-
464 chine learning), text mining is typically less concerned with linguistic structure, and more
465 interested in fast, scalable algorithms.

466 1.2 Three themes in natural language processing

467 Natural language processing covers a diverse range of tasks, methods, and linguistic phe-
468 nomena. But despite the apparent incommensurability between, say, the summarization
469 of scientific articles (§ 16.3.4.1) and the identification of suffix patterns in Spanish verbs
470 (§ 9.1.4.3), some general themes emerge. Each of these themes can be expressed as an
471 opposition between two extreme viewpoints on how to process natural language, and in
472 each case, existing approaches can be placed on a continuum between these two extremes.

473 1.2.1 Learning and knowledge

474 A recurring topic of debate is the relative importance of machine learning and linguistic
475 knowledge. On one extreme, advocates of “natural language processing from scratch” (Col-
476 lobert et al., 2011) propose to use machine learning to train end-to-end systems that trans-
477 mute raw text into any desired output structure: e.g., a summary, database, or transla-
478 tion. On the other extreme, the core work of natural language processing is sometimes
479 taken to be transforming text into a stack of general-purpose linguistic structures: from
480 subword units called **morphemes**, to word-level **parts-of-speech**, to tree-structured repre-
481 sentations of grammar, and beyond, to logic-based representations of meaning. In theory,
482 these general-purpose structures should then be able to support any desired application.

483 The end-to-end learning approach has been buoyed by recent results in computer vi-
484 sion and speech recognition, in which advances in machine learning have swept away
485 expert-engineered representations based on the fundamentals of optics and phonology (Krizhevsky
486 et al., 2012; Graves and Jaitly, 2014). But while some amount of machine learning is an el-
487 ement of nearly every contemporary approach to natural language processing, linguistic

488 representations such as syntax trees have not yet gone the way of the visual edge detector
 489 or the auditory triphone. Linguists have argued for the existence of a “language faculty”
 490 in all human beings, which encodes a set of abstractions specially designed to facilitate
 491 the understanding and production of language. The argument for the existence of such
 492 a language faculty is based on the observation that children learn language faster and
 493 from fewer examples than would be reasonably possible, if language was learned from
 494 experience alone.³ Regardless of the cognitive validity of these arguments, it seems that
 495 linguistic structures are particularly important in scenarios where training data is limited.

496 Moving away from the extreme ends of the continuum, there are a number of ways in
 497 which knowledge and learning can be combined in natural language processing. Many
 498 supervised learning systems make use of carefully engineered **features**, which transform
 499 the data into a representation that can facilitate learning. For example, in a task like doc-
 500 ument classification, it may be useful to identify each word’s **stem**, so that a learning
 501 system can more easily generalize across related terms such as *whale*, *whales*, *whalers*, and
 502 *whaling*. This is particularly important in the many languages that exceed English in the
 503 complexity of the system of affixes that can attach to words. Such features could be ob-
 504 tained from a hand-crafted resource, like a dictionary that maps each word to a single
 505 root form. Alternatively, features can be obtained from the output of a general-purpose
 506 language processing system, such as a parser or part-of-speech tagger, which may itself
 507 be built on supervised machine learning.

508 Another synthesis of learning and knowledge is in model structure: building machine
 509 learning models whose architectures are inspired by linguistic theories. For example, the
 510 organization of sentences is often described as **compositional**, with meaning of larger
 511 units gradually constructed from the meaning of their smaller constituents. This idea
 512 can be built into the architecture of a deep neural network, which is then trained using
 513 contemporary deep learning techniques (Dyer et al., 2016).

514 The debate about the relative importance of machine learning and linguistic knowl-
 515 edge sometimes becomes heated. No machine learning specialist likes to be told that their
 516 engineering methodology is unscientific alchemy;⁴ nor does a linguist want to hear that
 517 the search for general linguistic principles and structures has been made irrelevant by big
 518 data. Yet there is clearly room for both types of research: we need to know how far we
 519 can go with end-to-end learning alone, while at the same time, we continue the search for
 520 linguistic representations that generalize across applications, scenarios, and languages.
 521 For more on the history of this debate, see Church (2011); for an optimistic view of the
 522 potential symbiosis between computational linguistics and deep learning, see Manning

³The *Language Instinct* (Pinker, 2003) articulates these arguments in an engaging and popular style. For arguments against the innateness of language, see Elman et al. (1998).

⁴Ali Rahimi argued that much of deep learning research was similar to “alchemy” in a presentation at the 2017 conference on Neural Information Processing Systems. He was advocating for more learning theory, not more linguistics.

523 (2015).

524 **1.2.2 Search and learning**

525 Many natural language processing problems can be written mathematically in the form
 526 of optimization,⁵

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{x})} \Psi(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}), \quad [1.1]$$

527 where,

- 528 • \mathbf{x} is the input, which is an element of a set \mathcal{X} ;
- 529 • \mathbf{y} is the output, which is an element of a set $\mathcal{Y}(\mathbf{x})$;
- 530 • Ψ is a scoring function (also called the **model**), which maps from the set $\mathcal{X} \times \mathcal{Y}$ to
 531 the real numbers;
- 532 • $\boldsymbol{\theta}$ is a vector of parameters for Ψ ;
- 533 • $\hat{\mathbf{y}}$ is the predicted output, which is chosen to maximize the scoring function.

534 This basic structure can be used across a huge range of problems. For example, the
 535 input \mathbf{x} might be a social media post, and the output \mathbf{y} might be a labeling of the emotional
 536 sentiment expressed by the author (chapter 4); or \mathbf{x} could be a sentence in French, and the
 537 output \mathbf{y} could be a sentence in Tamil (chapter 18); or \mathbf{x} might be a sentence in English,
 538 and \mathbf{y} might be a representation of the syntactic structure of the sentence (chapter 10); or
 539 \mathbf{x} might be a news article and \mathbf{y} might be a structured record of the events that the article
 540 describes (chapter 17).

541 By adopting this formulation, we make an implicit decision that language processing
 542 algorithms will have two distinct modules:

543 **Search.** The search module is responsible for computing the argmax of the function Ψ . In
 544 other words, it finds the output $\hat{\mathbf{y}}$ that gets the best score with respect to the input
 545 \mathbf{x} . This is easy when the search space $\mathcal{Y}(\mathbf{x})$ is small enough to enumerate, or when
 546 the scoring function Ψ has a convenient decomposition into parts. In many cases,
 547 we will want to work with scoring functions that do not have these properties, moti-
 548 vating the use of more sophisticated search algorithms. Because the outputs are
 549 usually discrete in language processing problems, search often relies on the machin-
 550 ery of **combinatorial optimization**.

⁵Throughout this text, equations will be numbered by square brackets, and linguistic examples will be numbered by parentheses.

551 **Learning.** The learning module is responsible for finding the parameters θ . This is typ-
 552 ically (but not always) done by processing a large dataset of labeled examples,
 553 $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$. Like search, learning is also approached through the framework
 554 of optimization, as we will see in chapter 2. Because the parameters are usually
 555 continuous, learning algorithms generally rely on **numerical optimization**, search-
 556 ing over vectors of real numbers for parameters that optimize some function of the
 557 model and the labeled data. Some basic principles of numerical optimization are
 558 reviewed in Appendix B.

559 The division of natural language processing into separate modules for search and
 560 learning makes it possible to reuse generic algorithms across a range of different tasks
 561 and models. This means that the work of natural language processing can be focused on
 562 the design of the model Ψ , while reaping the benefits of decades of progress in search,
 563 optimization, and learning. Much of this textbook will focus on specific classes of scoring
 564 functions, and on the algorithms that make it possible to search and learn efficiently with
 565 them.

566 When a model is capable of making subtle linguistic distinctions, it is said to be *expres-*
 567 *sive*. Expressiveness is often traded off against the efficiency of search and learning. For
 568 example, a word-to-word translation model makes search and learning easy, but it is not
 569 expressive enough to distinguish good translations from bad ones. Unfortunately many
 570 of the most important problems in natural language processing seem to require expres-
 571 sive models, in which the complexity of search grows exponentially with the size of the
 572 input. In these models, exact search is usually impossible. Intractability threatens the neat
 573 modular decomposition between search and learning: if search requires a set of heuristic
 574 approximations, then it may be advantageous to learn a model that performs well under
 575 these specific heuristics. This has motivated some researchers to take a more integrated
 576 approach to search and learning, as briefly mentioned in chapters 11 and 15.

577 1.2.3 Relational, compositional, and distributional perspectives

578 Any element of language — a word, a phrase, a sentence, or even a sound — can be
 579 described from at least three perspectives. Consider the word *journalist*. A *journalist* is a
 580 subcategory of a *profession*, and an *anchorwoman* is a subcategory of *journalist*; furthermore,
 581 a *journalist* performs *journalism*, which is often, but not always, a subcategory of *writing*.
 582 This relational perspective on meaning is the basis for semantic **ontologies** such as **Word-**
 583 **Net** (Fellbaum, 2010), which enumerate the relations that hold between words and other
 584 elementary semantic units. The power of the relational perspective is illustrated by the
 585 following example:

586 (1.3) Umashanthi interviewed Ana. She works for the college newspaper.

587 Who works for the college newspaper? The word *journalist*, while not stated in the ex-
588 ample, implicitly links the *interview* to the *newspaper*, making *Umashanthi* the most likely
589 referent for the pronoun. (A general discussion of how to resolve pronouns is found in
590 chapter 15.)

591 Yet despite the inferential power of the relational perspective, it is not easy to formalize
592 computationally. Exactly which elements are to be related? Are *journalists* and *reporters*
593 distinct, or should we group them into a single unit? Is the kind of *interview* performed by
594 a journalist the same as the kind that one undergoes when applying for a job? Ontology
595 designers face many such thorny questions, and the project of ontology design hearkens
596 back to Borges' (1993) *Celestial Emporium of Benevolent Knowledge*, which divides animals
597 into:

598 (a) belonging to the emperor; (b) embalmed; (c) tame; (d) suckling pigs; (e)
599 sirens; (f) fabulous; (g) stray dogs; (h) included in the present classification;
600 (i) frenzied; (j) innumerable; (k) drawn with a very fine camelhair brush; (l) et
601 cetera; (m) having just broken the water pitcher; (n) that from a long way off
602 resemble flies.

603 Difficulties in ontology construction have led some linguists to argue that there is no task-
604 independent way to partition up word meanings (Kilgarriff, 1997).

605 Some problems are easier. Each member in a group of *journalists* is a *journalist*: the -s
606 suffix distinguishes the plural meaning from the singular in most of the nouns in English.
607 Similarly, a *journalist* can be thought of, perhaps colloquially, as someone who produces or
608 works on a *journal*. (Taking this approach even further, the word *journal* derives from the
609 French *jour+nal*, or *day+ly* = *daily*.) In this way, the meaning of a word is constructed from
610 the constituent parts — the principle of **compositionality**. This principle can be applied
611 to larger units: phrases, sentences, and beyond. Indeed, one of the great strengths of the
612 compositional view of meaning is that it provides a roadmap for understanding entire
613 texts and dialogues through a single analytic lens, grounding out in the smallest parts of
614 individual words.

615 But alongside *journalists* and *anti-parliamentarians*, there are many words that seem to
616 be linguistic atoms: think, for example, of *whale*, *blubber*, and *Nantucket*. Furthermore,
617 idiomatic phrases like *kick the bucket* and *shoot the breeze* have meanings that are quite
618 different from the sum of their parts (Sag et al., 2002). Composition is of little help for such
619 words and expressions, but their meanings can be ascertained — or at least approximated
620 — from the contexts in which they appear. Take, for example, *blubber*, which appears in
621 such contexts as:

- 622 (1.4) The blubber served them as fuel.
623 (1.5) ... extracting it from the blubber of the large fish ...

624 (1.6) Amongst oily substances, blubber has been employed as a manure.

625 These contexts form the **distributional properties** of the word *blubber*, and they link it to
 626 words which can appear in similar constructions: *fat*, *pelts*, and *barnacles*. This distribu-
 627 tional perspective makes it possible to learn about meaning from unlabeled data alone;
 628 unlike relational and compositional semantics, no manual annotation or expert knowl-
 629 edge is required. Distributional semantics is thus capable of covering a huge range of
 630 linguistic phenomena. However, it lacks precision: *blubber* is similar to *fat* in one sense, to
 631 *pelts* in another sense, and to *barnacles* in still another. The question of *why* all these words
 632 tend to appear in the same contexts is left unanswered.

633 The relational, compositional, and distributional perspectives all contribute to our un-
 634 derstanding of linguistic meaning, and all three appear to be critical to natural language
 635 processing. Yet they are uneasy collaborators, requiring seemingly incompatible repre-
 636 sentations and algorithmic approaches. This text presents some of the best known and
 637 most successful methods for working with each of these representations, but it is hoped
 638 that future research will reveal new ways to combine them.

639 1.3 Learning to do natural language processing

640 This text began with the notes that I use for teaching Georgia Tech’s undergraduate and
 641 graduate courses on natural language processing, CS 4650 and 7650. There are several
 642 other good resources (e.g., Manning and Schütze, 1999; Jurafsky and Martin, 2009; Smith,
 643 2011; Collins, 2013), but the goal of this text is focus on a core subset of the field, uni-
 644 fied by the concepts of learning and search. A remarkable thing about natural language
 645 processing is that so many problems can be solved by a compact set of methods:

646 **Search.** Viterbi, CKY, minimum spanning tree, shift-reduce, integer linear programming,
 647 beam search.

648 **Learning.** Naïve Bayes, logistic regression, perceptron, expectation-maximization, matrix
 649 factorization, backpropagation, recurrent neural networks.

650 This text explains how these methods work, and how they can be applied to problems
 651 that arise in the computer processing of natural language: document classification, word
 652 sense disambiguation, sequence labeling (part-of-speech tagging and named entity recog-
 653 nition), parsing, coreference resolution, relation extraction, discourse analysis, language
 654 modeling, and machine translation.

655 1.3.1 Background

656 Because natural language processing draws on many different intellectual traditions, al-
 657 most everyone who approaches it feels underprepared in one way or another. Here is a

658 summary of what is expected, and where you can learn more:

659 **Mathematics and machine learning.** The text assumes a background in multivariate cal-
660 culus and linear algebra: vectors, matrices, derivatives, and partial derivatives. You
661 should also be familiar with probability and statistics. A review of basic proba-
662 bility is found in Appendix A, and a minimal review of numerical optimization is
663 found in Appendix B. For linear algebra, the online course and textbook from Strang
664 (2016) are excellent sources of review material. Deisenroth et al. (2018) are currently
665 preparing a textbook on *Mathematics for Machine Learning*, and several chapters can
666 be found online.⁶ For an introduction to probabilistic modeling and estimation, see
667 James et al. (2013); for a more advanced and comprehensive discussion of the same
668 material, the classic reference is Hastie et al. (2009).

669 **Linguistics.** This book assumes no formal training in linguistics, aside from elementary
670 concepts like nouns and verbs, which you have probably encountered in the study
671 of English grammar. Ideas from linguistics are introduced throughout the text as
672 needed, including discussions of morphology and syntax (chapter 9), semantics
673 (chapters 12 and 13), and discourse (chapter 16). Linguistic issues also arise in the
674 application-focused chapters 4, 8, and 18. A short guide to linguistics for students
675 of natural language processing is offered by Bender (2013); you are encouraged to
676 start there, and then pick up a more comprehensive introductory textbook (e.g., Ak-
677 majian et al., 2010; Fromkin et al., 2013).

678 **Computer science.** The book is targeted at computer scientists, who are assumed to have
679 taken introductory courses on the analysis of algorithms and complexity theory. In
680 particular, you should be familiar with asymptotic analysis of the time and memory
681 costs of algorithms, and should have seen dynamic programming. The classic text
682 on algorithms is offered by Cormen et al. (2009); for an introduction to the theory of
683 computation, see Arora and Barak (2009) and Sipser (2012).

684 1.3.2 How to use this book

685 The textbook is organized into four main units:

686 **Learning.** This section builds up a set of machine learning tools that will be used through-
687 out the rest of the textbook. Because the focus is on machine learning, the text
688 representations and linguistic phenomena are mostly simple: “bag-of-words” text
689 classification is treated as a model example. Chapter 4 describes some of the more
690 linguistically interesting applications of word-based text analysis.

⁶<https://mml-book.github.io/>

691 **Sequences and trees.** This section introduces the treatment of language as a structured
 692 phenomena. It describes sequence and tree representations and the algorithms that
 693 they facilitate, as well as the limitations that these representations impose. Chapter
 694 9 introduces finite state automata and briefly overviews a context-free account of
 695 English syntax.

696 **Meaning.** This section takes a broad view of efforts to represent and compute meaning
 697 from text, ranging from formal logic to neural word embeddings. It also includes
 698 two topics that are closely related to semantics: resolution of ambiguous references,
 699 and analysis of multi-sentence discourse structure.

700 **Applications.** The final section offers chapter-length treatments on three of the most prominent
 701 applications of natural language processing: information extraction, machine
 702 translation, and text generation. Each of these applications merits a textbook length
 703 treatment of its own (Koehn, 2009; Grishman, 2012; Reiter and Dale, 2000); the chapters
 704 here explain some of the most well known systems using the formalisms and
 705 methods built up earlier in the book, while introducing methods such as neural attention.
 706

707 Each chapter contains some advanced material, which is marked with an asterisk.
 708 This material can be safely omitted without causing misunderstandings later on. But
 709 even without these advanced sections, the text is too long for a single semester course, so
 710 instructors will have to pick and choose among the chapters.

711 Chapters 2 and 3 provide building blocks that will be used throughout the book, and
 712 chapter 4 describes some critical aspects of the practice of language technology. Lan-
 713 guage models (chapter 6), sequence labeling (chapter 7), and parsing (chapter 10 and 11)
 714 are canonical topics in natural language processing, and distributed word embeddings
 715 (chapter 14) are so ubiquitous that students will complain if you leave them out. Of the
 716 applications, machine translation (chapter 18) is the best choice: it is more cohesive than
 717 information extraction, and more mature than text generation. In my experience, nearly
 718 all students benefit from the review of probability in Appendix A.

- 719 • A course focusing on machine learning should add the chapter on unsupervised
 720 learning (chapter 5). The chapters on predicate-argument semantics (chapter 13),
 721 reference resolution (chapter 15), and text generation (chapter 19) are particularly
 722 influenced by recent machine learning innovations, including deep neural networks
 723 and learning to search.
- 724 • A course with a more linguistic orientation should add the chapters on applica-
 725 tions of sequence labeling (chapter 8), formal language theory (chapter 9), semantics
 726 (chapter 12 and 13), and discourse (chapter 16).

- 727 • For a course with a more applied focus — for example, a course targeting under-
728 graduates — I recommend the chapters on applications of sequence labeling (chap-
729 ter 8), predicate-argument semantics (chapter 13), information extraction (chapter 17),
730 and text generation (chapter 19).

731 **Acknowledgments**

732 Several of my colleagues and students read early drafts of chapters in their areas of exper-
733 tise, including Yoav Artzi, Kevin Duh, Heng Ji, Jessy Li, Brendan O'Connor, Yuval Pin-
734 ter, Nathan Schneider, Pamela Shapiro, Noah A. Smith, Sandeep Soni, and Luke Zettle-
735 moyer. I would also like to thank the following people for helpful discussions of the
736 material: Kevin Murphy, Shawn Ling Ramirez, William Yang Wang, and Bonnie Webber.
737 Several students, colleagues, friends, and family found mistakes in early drafts: Parmin-
738 der Bhatia, Kimberly Caras, Murtaza Dhuliawala, Barbara Eisenstein, Chris Gu, Joshua
739 Killingsworth, Jonathan May, Taha Merghani, Gus Monod, Raghavendra Murali, Nidish
740 Nair, Brendan O'Connor, Yuval Pinter, Nathan Schneider, Zhewei Sun, Ashwin Cunna-
741 pakkam Vinjimir, Clay Washington, Ishan Waykul, and Yuyu Zhang. Special thanks to
742 the many students in Georgia Tech's CS 4650 and 7650 who suffered through early ver-
743 sions of the text.

744

Part I

745

Learning

746

Chapter 2

747

Linear text classification

748 We'll start with the problem of **text classification**: given a text document, assign it a dis-
749 crete label $y \in \mathcal{Y}$, where \mathcal{Y} is the set of possible labels. This problem has many appli-
750 cations, from spam filtering to analysis of electronic health records. Text classification is
751 also a building block that is used throughout more complex natural language processing
752 tasks.

753 To perform this task, the first question is how to represent each document. A common
754 approach is to use a vector of word counts, e.g., $\mathbf{x} = [0, 1, 1, 0, 0, 2, 0, 1, 13, 0 \dots]^T$, where
755 x_j is the count of word j . The length of \mathbf{x} is $V \triangleq |\mathcal{V}|$, where \mathcal{V} is the set of possible words
756 in the vocabulary.

757 The object \mathbf{x} is a vector, but colloquially we call it a **bag of words**, because it includes
758 only information about the count of each word, and not the order in which the words
759 appear. We have thrown out grammar, sentence boundaries, paragraphs — everything
760 but the words. Yet the bag of words model is surprisingly effective for text classification.
761 If you see the word *freeee* in an email, is it a spam email? What if you see the word
762 *Bayesian*? For many labeling problems, individual words can be strong predictors.

763 To predict a label from a bag-of-words, we can assign a score to each word in the
764 vocabulary, measuring the compatibility with the label. In the spam filtering case, we
765 might assign a positive score to the word *freeee* for the label SPAM, and a negative score
766 to the word *Bayesian*. These scores are called **weights**, and they are arranged in a column
767 vector θ .

768 Suppose that you want a multiclass classifier, where $K \triangleq |\mathcal{Y}| > 2$. For example, we
769 might want to classify news stories about sports, celebrities, music, and business. The goal
770 is to predict a label \hat{y} , given the bag of words \mathbf{x} , using the weights θ . For each label $y \in \mathcal{Y}$,
771 we compute a score $\Psi(\mathbf{x}, y)$, which is a scalar measure of the compatibility between the
772 bag-of-words \mathbf{x} and the label y . In a linear bag-of-words classifier, this score is the vector

773 inner product between the weights θ and the output of a **feature function** $f(x, y)$,

$$\Psi(x, y) = \theta \cdot f(x, y). \quad [2.1]$$

774 As the notation suggests, f is a function of two arguments, the word counts x and the
 775 label y , and it returns a vector output. For example, given arguments x and y , element j
 776 of this feature vector might be,

$$f_j(x, y) = \begin{cases} x_{freeee}, & \text{if } y = \text{SPAM} \\ 0, & \text{otherwise} \end{cases} \quad [2.2]$$

777 This function returns the count of the word *freeee* if the label is SPAM, and it returns zero
 778 otherwise. The corresponding weight θ_j then scores the compatibility of the word *freeee*
 779 with the label SPAM. A positive score means that this word makes the label more likely.

To formalize this feature function, we define $f(x, y)$ as a column vector,

$$f(x, y = 1) = [x; \underbrace{0; 0; \dots; 0}_{(K-1) \times V}] \quad [2.3]$$

$$f(x, y = 2) = [\underbrace{0; 0; \dots; 0}_V; x; \underbrace{0; 0; \dots; 0}_{(K-2) \times V}] \quad [2.4]$$

$$f(x, y = K) = [\underbrace{0; 0; \dots; 0}_{(K-1) \times V}; x], \quad [2.5]$$

780 where $\underbrace{[0; 0; \dots; 0]}_{(K-1) \times V}$ is a column vector of $(K - 1) \times V$ zeros, and the semicolon indicates
 781 vertical concatenation. This arrangement is shown in Figure 2.1; the notation may seem
 782 awkward at first, but it generalizes to an impressive range of learning settings.

Given a vector of weights, $\theta \in \mathbb{R}^{V \times K}$, we can now compute the score $\Psi(x, y)$. This
 inner product gives a scalar measure of the compatibility of the observation x with label
 y .¹ For any document x , we predict the label \hat{y} ,

$$\hat{y} = \operatorname{argmax}_{y \in \mathcal{Y}} \Psi(x, y) \quad [2.6]$$

$$\Psi(x, y) = \theta \cdot f(x, y). \quad [2.7]$$

783 This inner product notation gives a clean separation between the *data* (x and y) and the
 784 *parameters* (θ). This notation also generalizes nicely to **structured prediction**, in which

¹Only $V \times (K - 1)$ features and weights are necessary. By stipulating that $\Psi(x, y = K) = 0$ regardless of x , it is possible to implement any classification rule that can be achieved with $V \times K$ features and weights. This is the approach taken in binary classification rules like $y = \text{Sign}(\beta \cdot x + a)$, where β is a vector of weights, a is an offset, and the label set is $\mathcal{Y} = \{-1, 1\}$. However, for multiclass classification, it is more concise to write $\theta \cdot f(x, y)$ for all $y \in \mathcal{Y}$.

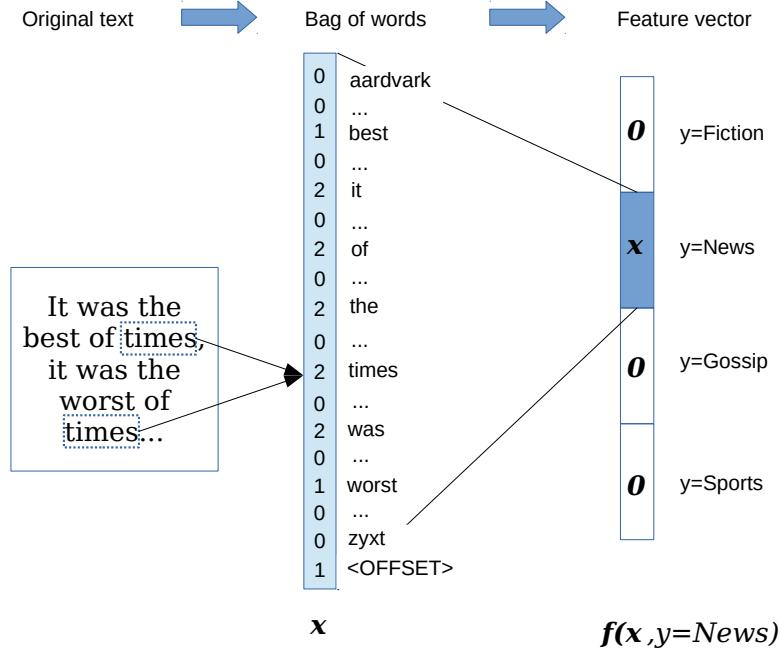


Figure 2.1: The bag-of-words and feature vector representations, for a hypothetical text classification task.

785 the space of labels \mathcal{Y} is very large, and we want to model shared substructures between
786 labels.

787 It is common to add an **offset feature** at the end of the vector of word counts x , which
788 is always 1. We then have to also add an extra zero to each of the zero vectors, to make the
789 vector lengths match. This gives the entire feature vector $f(x, y)$ a length of $(V + 1) \times K$.
790 The weight associated with this offset feature can be thought of as a bias for or against
791 each label. For example, if we expect most documents to be spam, then the weight for
792 the offset feature for $y = \text{SPAM}$ should be larger than the weight for the offset feature for
793 $y = \text{HAM}$.

Returning to the weights θ , where do they come from? One possibility is to set them by hand. If we wanted to distinguish, say, English from Spanish, we can use English and Spanish dictionaries, and set the weight to one for each word that appears in the

associated dictionary. For example,²

$$\begin{array}{ll} \theta_{(E,bicycle)} = 1 & \theta_{(S,bicycle)} = 0 \\ \theta_{(E,bicicleta)} = 0 & \theta_{(S,bicicleta)} = 1 \\ \theta_{(E,con)} = 1 & \theta_{(S,con)} = 1 \\ \theta_{(E,ordinateur)} = 0 & \theta_{(S,ordinateur)} = 0. \end{array}$$

794 Similarly, if we want to distinguish positive and negative sentiment, we could use positive
795 and negative **sentiment lexicons** (see § 4.1.2), which are defined by social psychologists
796 (Tausczik and Pennebaker, 2010).

797 But it is usually not easy to set classification weights by hand, due to the large number
798 of words and the difficulty of selecting exact numerical weights. Instead, we will learn the
799 weights from data. Email users manually label messages as SPAM; newspapers label their
800 own articles as BUSINESS or STYLE. Using such **instance labels**, we can automatically
801 acquire weights using **supervised machine learning**. This chapter will discuss several
802 machine learning approaches for classification. The first is based on probability. For a
803 review of probability, consult Appendix A.

804 2.1 Naïve Bayes

805 The **joint probability** of a bag of words \mathbf{x} and its true label y is written $p(\mathbf{x}, y)$. Suppose
806 we have a dataset of N labeled instances, $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$, which we assume are **independ-**
807 **ent and identically distributed (IID)** (see § A.3). Then the joint probability of the entire
808 dataset, written $p(\mathbf{x}^{(1:N)}, y^{(1:N)})$, is equal to $\prod_{i=1}^N p_{X,Y}(\mathbf{x}^{(i)}, y^{(i)})$.³

What does this have to do with classification? One approach to classification is to set the weights $\boldsymbol{\theta}$ so as to maximize the joint probability of a **training set** of labeled documents. This is known as **maximum likelihood estimation**:

$$\hat{\boldsymbol{\theta}} = \operatorname{argmax}_{\boldsymbol{\theta}} p(\mathbf{x}^{(1:N)}, y^{(1:N)}; \boldsymbol{\theta}) \quad [2.8]$$

$$= \operatorname{argmax}_{\boldsymbol{\theta}} \prod_{i=1}^N p(\mathbf{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}) \quad [2.9]$$

$$= \operatorname{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^N \log p(\mathbf{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}). \quad [2.10]$$

²In this notation, each tuple (language, word) indexes an element in $\boldsymbol{\theta}$, which remains a vector.

³The notation $p_{X,Y}(\mathbf{x}^{(i)}, y^{(i)})$ indicates the joint probability that random variables X and Y take the specific values $\mathbf{x}^{(i)}$ and $y^{(i)}$ respectively. The subscript will often be omitted when it is clear from context. For a review of random variables, see Appendix A.

Algorithm 1 Generative process for the Naïve Bayes classifier

for Document $i \in \{1, 2, \dots, N\}$ **do**:

Draw the label $y^{(i)} \sim \text{Categorical}(\boldsymbol{\mu})$;

Draw the word counts $\mathbf{x}^{(i)} | y^{(i)} \sim \text{Multinomial}(\boldsymbol{\phi}_{y^{(i)}})$.

809 The notation $p(\mathbf{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})$ indicates that $\boldsymbol{\theta}$ is a *parameter* of the probability function. The
 810 product of probabilities can be replaced by a sum of log-probabilities because the log func-
 811 tion is monotonically increasing over positive arguments, and so the same $\boldsymbol{\theta}$ will maxi-
 812 mize both the probability and its logarithm. Working with logarithms is desirable because
 813 of numerical stability: on a large dataset, multiplying many probabilities can **underflow**
 814 to zero.⁴

815 The probability $p(\mathbf{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})$ is defined through a **generative model** — an idealized
 816 random process that has generated the observed data.⁵ Algorithm 1 describes the gener-
 817 ative model describes the **Naïve Bayes** classifier, with parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}, \boldsymbol{\phi}\}$.

- 818 • The first line of this generative model encodes the assumption that the instances are
 819 mutually independent: neither the label nor the text of document i affects the label
 820 or text of document j .⁶ Furthermore, the instances are identically distributed: the
 821 distributions over the label $y^{(i)}$ and the text $\mathbf{x}^{(i)}$ (conditioned on $y^{(i)}$) are the same
 822 for all instances i .
- 823 • The second line of the generative model states that the random variable $y^{(i)}$ is drawn
 824 from a categorical distribution with parameter $\boldsymbol{\mu}$. Categorical distributions are like
 825 weighted dice: the vector $\boldsymbol{\mu} = [\mu_1, \mu_2, \dots, \mu_K]^\top$ gives the probabilities of each la-
 826 bel, so that the probability of drawing label y is equal to μ_y . For example, if $\mathcal{Y} =$
 827 $\{\text{POSITIVE}, \text{NEGATIVE}, \text{NEUTRAL}\}$, we might have $\boldsymbol{\mu} = [0.1, 0.7, 0.2]^\top$. We require
 828 $\sum_{y \in \mathcal{Y}} \mu_y = 1$ and $\mu_y \geq 0, \forall y \in \mathcal{Y}$.⁷
- 829 • The third line describes how the bag-of-words counts $\mathbf{x}^{(i)}$ are generated. By writing
 830 $\mathbf{x}^{(i)} | y^{(i)}$, this line indicates that the word counts are conditioned on the label, so

⁴Throughout this text, you may assume all logarithms and exponents are base 2, unless otherwise indicated. Any reasonable base will yield an identical classifier, and base 2 is most convenient for working out examples by hand.

⁵Generative models will be used throughout this text. They explicitly define the assumptions underlying the form of a probability distribution over observed and latent variables. For a readable introduction to generative models in statistics, see Blei (2014).

⁶Can you think of any cases in which this assumption is too strong?

⁷Formally, we require $\boldsymbol{\mu} \in \Delta^{K-1}$, where Δ^{K-1} is the $K - 1$ **probability simplex**, the set of all vectors of K nonnegative numbers that sum to one. Because of the sum-to-one constraint, there are $K - 1$ degrees of freedom for a vector of size K .

831 that the joint probability is factored using the chain rule,

$$p_{X,Y}(x^{(i)}, y^{(i)}) = p_{X|Y}(x^{(i)} | y^{(i)}) \times p_Y(y^{(i)}). \quad [2.11]$$

The specific distribution $p_{X|Y}$ is the **multinomial**, which is a probability distribution over vectors of non-negative counts. The probability mass function for this distribution is:

$$p_{\text{mult}}(\mathbf{x}; \boldsymbol{\phi}) = B(\mathbf{x}) \prod_{j=1}^V \phi_j^{x_j} \quad [2.12]$$

$$B(\mathbf{x}) = \frac{(\sum_{j=1}^V x_j)!}{\prod_{j=1}^V (x_j)!} \quad [2.13]$$

832 As in the categorical distribution, the parameter ϕ_j can be interpreted as a proba-
 833 bility: specifically, the probability that any given token in the document is the word
 834 j . The multinomial distribution involves a product over words, with each term in
 835 the product equal to the probability ϕ_j , exponentiated by the count x_j . Words that
 836 have zero count play no role in this product, because $\phi_j^0 = 1$. The term $B(\mathbf{x})$ doesn't
 837 depend on $\boldsymbol{\phi}$, and can usually be ignored. Can you see why we need this term at
 838 all?⁸

839 The notation $p(\mathbf{x} | y; \boldsymbol{\phi})$ indicates the conditional probability of word counts \mathbf{x} given
 840 label y , with parameter $\boldsymbol{\phi}$, which is equal to $p_{\text{mult}}(\mathbf{x}; \boldsymbol{\phi}_y)$. By specifying the multino-
 841 mial distribution, we describe the **multinomial naïve Bayes** classifier. Why “naïve”?
 842 Because the multinomial distribution treats each word token independently: the
 843 probability mass function factorizes across the counts.⁹

844 2.1.1 Types and tokens

845 A slight modification to the generative model of Naïve Bayes is shown in Algorithm 2.
 846 Instead of generating a vector of counts of **types**, \mathbf{x} , this model generates a *sequence of*
 847 **tokens**, $\mathbf{w} = (w_1, w_2, \dots, w_M)$. The distinction between types and tokens is critical: $x_j \in$
 848 $\{0, 1, 2, \dots, M\}$ is the count of word type j in the vocabulary, e.g., the number of times
 849 the word *cannibal* appears; $w_m \in \mathcal{V}$ is the identity of token m in the document, e.g. $w_m =$
 850 *cannibal*.

⁸Technically, a multinomial distribution requires a second parameter, the total number of word counts in \mathbf{x} . In the bag-of-words representation is equal to the number of words in the document. However, this parameter is irrelevant for classification.

⁹You can plug in any probability distribution to the generative story and it will still be Naïve Bayes, as long as you are making the “naïve” assumption that the features are conditionally independent, given the label. For example, a multivariate Gaussian with diagonal covariance is naïve in exactly the same sense.

Algorithm 2 Alternative generative process for the Naïve Bayes classifier

```

for Document  $i \in \{1, 2, \dots, N\}$  do:
    Draw the label  $y^{(i)} \sim \text{Categorical}(\boldsymbol{\mu})$ ;
    for Token  $m \in \{1, 2, \dots, M_i\}$  do:
        Draw the token  $w_m^{(i)} | y^{(i)} \sim \text{Categorical}(\boldsymbol{\phi}_{y^{(i)}})$ .

```

851 The probability of the sequence \mathbf{w} is a product of categorical probabilities. Algo-
 852 rithm 2 makes a conditional independence assumption: each token $w_m^{(i)}$ is independent
 853 of all other tokens $w_{n \neq m}^{(i)}$, conditioned on the label $y^{(i)}$. This is identical to the “naïve”
 854 independence assumption implied by the multinomial distribution, and as a result, the
 855 optimal parameters for this model are identical to those in multinomial Naïve Bayes. For
 856 any instance, the probability assigned by this model is proportional to the probability un-
 857 der multinomial Naïve Bayes. The constant of proportionality is the factor $B(\mathbf{x})$, which
 858 appears in the multinomial distribution. Because $B(\mathbf{x}) \geq 1$, the probability for a vector
 859 of counts \mathbf{x} is at least as large as the probability for a list of words \mathbf{w} that induces the
 860 same counts: there can be many word sequences that correspond to a single vector of
 861 counts. For example, *man bites dog* and *dog bites man* correspond to an identical count vec-
 862 tor, $\{bites : 1, dog : 1, man : 1\}$, and $B(\mathbf{x})$ is equal to the total number of possible word
 863 orderings for count vector \mathbf{x} .

864 Sometimes it is useful to think of instances as counts of types, \mathbf{x} ; other times, it is
 865 better to think of them as sequences of tokens, \mathbf{w} . If the tokens are generated from a
 866 model that assumes conditional independence, then these two views lead to probability
 867 models that are identical, except for a scaling factor that does not depend on the label or
 868 the parameters.

869 **2.1.2 Prediction**

The Naïve Bayes prediction rule is to choose the label y which maximizes $\log p(\mathbf{x}, y; \boldsymbol{\mu}, \boldsymbol{\phi})$:

$$\hat{y} = \underset{y}{\operatorname{argmax}} \log p(\mathbf{x}, y; \boldsymbol{\mu}, \boldsymbol{\phi}) \quad [2.14]$$

$$= \underset{y}{\operatorname{argmax}} \log p(\mathbf{x} | y; \boldsymbol{\phi}) + \log p(y; \boldsymbol{\mu}) \quad [2.15]$$

Now we can plug in the probability distributions from the generative story.

$$\log p(\mathbf{x} \mid y; \boldsymbol{\phi}) + \log p(y; \boldsymbol{\mu}) = \log \left[B(\mathbf{x}) \prod_{j=1}^V \phi_{y,j}^{x_j} \right] + \log \mu_y \quad [2.16]$$

$$= \log B(\mathbf{x}) + \sum_{j=1}^V x_j \log \phi_{y,j} + \log \mu_y \quad [2.17]$$

$$= \log B(\mathbf{x}) + \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y), \quad [2.18]$$

where

$$\boldsymbol{\theta} = [\boldsymbol{\theta}^{(1)}; \boldsymbol{\theta}^{(2)}; \dots; \boldsymbol{\theta}^{(K)}] \quad [2.19]$$

$$\boldsymbol{\theta}^{(y)} = [\log \phi_{y,1}; \log \phi_{y,2}; \dots; \log \phi_{y,V}; \log \mu_y] \quad [2.20]$$

870 The feature function $\mathbf{f}(\mathbf{x}, y)$ is a vector of V word counts and an offset, padded by
 871 zeros for the labels not equal to y (see Equations 2.3-2.5, and Figure 2.1). This construction
 872 ensures that the inner product $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y)$ only activates the features whose weights are
 873 in $\boldsymbol{\theta}^{(y)}$. These features and weights are all we need to compute the joint log-probability
 874 $\log p(\mathbf{x}, y)$ for each y . This is a key point: through this notation, we have converted the
 875 problem of computing the log-likelihood for a document-label pair (\mathbf{x}, y) into the compu-
 876 tation of a vector inner product.

877 2.1.3 Estimation

878 The parameters of the categorical and multinomial distributions have a simple interpre-
 879 tation: they are vectors of expected frequencies for each possible event. Based on this
 880 interpretation, it is tempting to set the parameters empirically,

$$\phi_{y,j} = \frac{\text{count}(y, j)}{\sum_{j'=1}^V \text{count}(y, j')} = \frac{\sum_{i:y^{(i)}=y} x_j^{(i)}}{\sum_{j'=1}^V \sum_{i:y^{(i)}=y} x_{j'}^{(i)}}, \quad [2.21]$$

881 where $\text{count}(y, j)$ refers to the count of word j in documents with label y .

882 Equation 2.21 defines the **relative frequency estimate** for ϕ . It can be justified as a
 883 **maximum likelihood estimate**: the estimate that maximizes the probability $p(\mathbf{x}^{(1:N)}, y^{(1:N)}; \boldsymbol{\theta})$.
 884 Based on the generative model in Algorithm 1, the log-likelihood is,

$$\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\mu}) = \sum_{i=1}^N \log p_{\text{mult}}(\mathbf{x}^{(i)}; \boldsymbol{\phi}_{y^{(i)}}) + \log p_{\text{cat}}(y^{(i)}; \boldsymbol{\mu}), \quad [2.22]$$

which is now written as a function \mathcal{L} of the parameters ϕ and μ . Let's continue to focus on the parameters ϕ . Since $p(y)$ is constant with respect to ϕ , we can drop it:

$$\mathcal{L}(\phi) = \sum_{i=1}^N \log p_{\text{mult}}(\mathbf{x}^{(i)}; \phi_{y^{(i)}}) = \sum_{i=1}^N \log B(\mathbf{x}^{(i)}) + \sum_{j=1}^V x_j^{(i)} \log \phi_{y^{(i)}, j}, \quad [2.23]$$

where $B(\mathbf{x}^{(i)})$ is constant with respect to ϕ .

We would now like to optimize the log-likelihood \mathcal{L} , by taking derivatives with respect to ϕ . But before we can do that, we have to deal with a set of constraints:

$$\sum_{j=1}^V \phi_{y,j} = 1 \quad \forall y \quad [2.24]$$

These constraints can be incorporated by adding a set of Lagrange multipliers (see Appendix B for more details). Solving separately for each label y , we obtain the Lagrangian,

$$\ell(\phi_y) = \sum_{i:y^{(i)}=y} \sum_{j=1}^V x_j^{(i)} \log \phi_{y,j} - \lambda \left(\sum_{j=1}^V \phi_{y,j} - 1 \right). \quad [2.25]$$

It is now possible to differentiate the Lagrangian with respect to the parameter of interest,

$$\frac{\partial \ell(\phi_y)}{\partial \phi_{y,j}} = \sum_{i:y^{(i)}=y} x_j^{(i)} / \phi_{y,j} - \lambda \quad [2.26]$$

The solution is obtained by setting each element in this vector of derivatives equal to zero,

$$\lambda \phi_{y,j} = \sum_{i:y^{(i)}=y} x_j^{(i)} \quad [2.27]$$

$$\phi_{y,j} \propto \sum_{i:y^{(i)}=y} x_j^{(i)} = \sum_{i=1}^N \delta(y^{(i)} = y) x_j^{(i)} = \text{count}(y, j), \quad [2.28]$$

where $\delta(y^{(i)} = y)$ is a **delta function**, also sometimes called an **indicator function**, which returns one if $y^{(i)} = y$, and zero otherwise. Equation 2.28 shows three different notations for the same thing: a sum over the word counts for all documents i such that the label $y^{(i)} = y$. This gives a solution for each ϕ_y up to a constant of proportionality. Now recall the constraint $\sum_{j=1}^V \phi_{y,j} = 1$, which arises because ϕ_y represents a vector of probabilities for each word in the vocabulary. This constraint leads to an exact solution,

$$\phi_{y,j} = \frac{\text{count}(y, j)}{\sum_{j'=1}^V \text{count}(y, j')}. \quad [2.29]$$

890 This is equal to the relative frequency estimator from Equation 2.21. A similar derivation
 891 gives $\mu_y \propto \sum_{i=1}^N \delta(y^{(i)} = y)$.

892 2.1.4 Smoothing and MAP estimation

893 With text data, there are likely to be pairs of labels and words that never appear in the
 894 training set, leaving $\phi_{y,j} = 0$. For example, the word *Bayesian* may have never yet ap-
 895 peared in a spam email. But choosing a value of $\phi_{\text{SPAM}, \text{Bayesian}} = 0$ would allow this single
 896 feature to completely veto a label, since $p(\text{SPAM} | x) = 0$ if $x_{\text{Bayesian}} > 0$.

897 This is undesirable, because it imposes high **variance**: depending on what data hap-
 898 pens to be in the training set, we could get vastly different classification rules. One so-
 899 lution is to **smooth** the probabilities, by adding a “pseudocount” of α to each count, and
 900 then normalizing.

$$\phi_{y,j} = \frac{\alpha + \text{count}(y, j)}{V\alpha + \sum_{j'=1}^V \text{count}(y, j')} \quad [2.30]$$

901 This is called **Laplace smoothing**.¹⁰ The pseudocount α is a **hyperparameter**, because it
 902 controls the form of the log-likelihood function, which in turn drives the estimation of ϕ .

903 Smoothing reduces variance, but it takes us away from the maximum likelihood esti-
 904 mate: it imposes a **bias**. In this case, the bias points towards uniform probabilities. Ma-
 905 chine learning theory shows that errors on heldout data can be attributed to the sum of
 906 bias and variance (Mohri et al., 2012). Techniques for reducing variance typically increase
 907 the bias, leading to a **bias-variance tradeoff**.

- 908 • Unbiased classifiers may **overfit** the training data, yielding poor performance on
 909 unseen data.
- 910 • But if the smoothing is too large, the resulting classifier can **underfit** instead. In the
 911 limit of $\alpha \rightarrow \infty$, there is zero variance: you get the same classifier, regardless of the
 912 data. However, the bias is likely to be large.

913 2.1.5 Setting hyperparameters

914 How should we choose the best value of hyperparameters like α ? Maximum likelihood
 915 will not work: the maximum likelihood estimate of α on the training set will always be
 916 $\alpha = 0$. In many cases, what we really want is **accuracy**: the number of correct predictions,
 917 divided by the total number of predictions. (Other measures of classification performance
 918 are discussed in § 4.4.) As we will see, it is hard to optimize for accuracy directly. But for
 919 scalar hyperparameters like α can be tuned by a simple heuristic called **grid search**: try a

¹⁰Laplace smoothing has a Bayesian justification, in which the generative model is extended to include ϕ as a random variable. The resulting estimate is called **maximum a posteriori**, or MAP.

920 set of values (e.g., $\alpha \in \{0.001, 0.01, 0.1, 1, 10\}$), compute the accuracy for each value, and
921 choose the setting that maximizes the accuracy.

922 The goal is to tune α so that the classifier performs well on *unseen* data. For this reason,
923 the data used for hyperparameter tuning should not overlap the training set, where very
924 small values of α will be preferred. Instead, we hold out a **development set** (also called
925 a **tuning set**) for hyperparameter selection. This development set may consist of a small
926 fraction of the labeled data, such as 10%.

927 We also want to predict the performance of our classifier on unseen data. To do this,
928 we must hold out a separate subset of data, called the **test set**. It is critical that the test set
929 not overlap with either the training or development sets, or else we will overestimate the
930 performance that the classifier will achieve on unlabeled data in the future. The test set
931 should also not be used when making modeling decisions, such as the form of the feature
932 function, the size of the vocabulary, and so on (these decisions are reviewed in chapter 4.)
933 The ideal practice is to use the test set only once — otherwise, the test set is used to guide
934 the classifier design, and test set accuracy will diverge from accuracy on truly unseen
935 data. Because annotated data is expensive, this ideal can be hard to follow in practice,
936 and many test sets have been used for decades. But in some high-impact applications like
937 machine translation and information extraction, new test sets are released every year.

938 When only a small amount of labeled data is available, the test set accuracy can be
939 unreliable. *K*-fold **cross-validation** is one way to cope with this scenario: the labeled
940 data is divided into *K* folds, and each fold acts as the test set, while training on the other
941 folds. The test set accuracies are then aggregated. In the extreme, each fold is a single data
942 point; this is called **leave-one-out** cross-validation. To perform hyperparameter tuning in
943 the context of cross-validation, another fold can be used for grid search. It is important
944 not to repeatedly evaluate the cross-validated accuracy while making design decisions
945 about the classifier, or you will overstate the accuracy on truly unseen data.

946 2.2 Discriminative learning

947 Naïve Bayes is easy to work with: the weights can be estimated in closed form, and the
948 probabilistic interpretation makes it relatively easy to extend. However, the assumption
949 that features are independent can seriously limit its accuracy. Thus far, we have defined
950 the **feature function** $f(\mathbf{x}, y)$ so that it corresponds to bag-of-words features: one feature
951 per word in the vocabulary. In natural language, bag-of-words features violate the as-
952 sumption of conditional independence — for example, the probability that a document
953 will contain the word *naïve* is surely higher given that it also contains the word *Bayes* —
954 but this violation is relatively mild.

955 However, good performance on text classification often requires features that are richer
956 than the bag-of-words:

- 957 • To better handle out-of-vocabulary terms, we want features that apply to multiple
 958 words, such as prefixes and suffixes (e.g., *anti-*, *un-*, *-ing*) and capitalization.
 959 • We also want *n-gram* features that apply to multi-word units: **bigrams** (e.g., *not*
 960 *good*, *not bad*), **trigrams** (e.g., *not so bad*, *lacking any decency*, *never before imagined*), and
 961 beyond.

These features flagrantly violate the Naïve Bayes independence assumption. Consider what happens if we add a prefix feature. Under the Naïve Bayes assumption, we make the following approximation:¹¹

$$\Pr(\text{word} = \textit{unfit}, \text{prefix} = \textit{un-} \mid y) \approx \Pr(\text{prefix} = \textit{un-} \mid y) \times \Pr(\text{word} = \textit{unfit} \mid y).$$

To test the quality of the approximation, we can manipulate the left-hand side by applying the chain rule,

$$\Pr(\text{word} = \textit{unfit}, \text{prefix} = \textit{un-} \mid y) = \Pr(\text{prefix} = \textit{un-} \mid \text{word} = \textit{unfit}, y) \quad [2.31]$$

$$\times \Pr(\text{word} = \textit{unfit} \mid y) \quad [2.32]$$

But $\Pr(\text{prefix} = \textit{un-} \mid \text{word} = \textit{unfit}, y) = 1$, since *un-* is guaranteed to be the prefix for the word *unfit*. Therefore,

$$\Pr(\text{word} = \textit{unfit}, \text{prefix} = \textit{un-} \mid y) = 1 \times \Pr(\text{word} = \textit{unfit} \mid y) \quad [2.33]$$

$$\gg \Pr(\text{prefix} = \textit{un-} \mid y) \times \Pr(\text{word} = \textit{unfit} \mid y), \quad [2.34]$$

962 because the probability of any given word starting with the prefix *un-* is much less than
 963 one. Naïve Bayes will systematically underestimate the true probabilities of conjunctions
 964 of positively correlated features. To use such features, we need learning algorithms that
 965 do not rely on an independence assumption.

966 The origin of the Naïve Bayes independence assumption is the learning objective,
 967 $p(\mathbf{x}^{(1:N)}, y^{(1:N)})$, which requires modeling the probability of the observed text. In clas-
 968 sification problems, we are always given \mathbf{x} , and are only interested in predicting the label
 969 y , so it seems unnecessary to model the probability of \mathbf{x} . **Discriminative learning** algo-
 970 rithms focus on the problem of predicting y , and do not attempt to model the probability
 971 of the text \mathbf{x} .

972 2.2.1 Perceptron

973 In Naïve Bayes, the weights can be interpreted as parameters of a probabilistic model. But
 974 this model requires an independence assumption that usually does not hold, and limits

¹¹The notation $\Pr(\cdot)$ refers to the probability of an event, and $p(\cdot)$ refers to the probability density or mass for a random variable (see Appendix A).

Algorithm 3 Perceptron learning algorithm

```

1: procedure PERCEPTRON( $\mathbf{x}^{(1:N)}, y^{(1:N)}$ )
2:    $t \leftarrow 0$ 
3:    $\boldsymbol{\theta}^{(0)} \leftarrow \mathbf{0}$ 
4:   repeat
5:      $t \leftarrow t + 1$ 
6:     Select an instance  $i$ 
7:      $\hat{y} \leftarrow \operatorname{argmax}_y \boldsymbol{\theta}^{(t-1)} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y)$ 
8:     if  $\hat{y} \neq y^{(i)}$  then
9:        $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} + \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - \mathbf{f}(\mathbf{x}^{(i)}, \hat{y})$ 
10:    else
11:       $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)}$ 
12:    until tired
13:   return  $\boldsymbol{\theta}^{(t)}$ 

```

975 our choice of features. Why not forget about probability and learn the weights in an error-
 976 driven way? The **perceptron** algorithm, shown in Algorithm 3, is one way to do this.

977 Here's what the algorithm says: if you make a mistake, increase the weights for fea-
 978 tures that are active with the correct label $y^{(i)}$, and decrease the weights for features that
 979 are active with the guessed label \hat{y} . This is an **online learning** algorithm, since the clas-
 980 sifier weights change after every example. This is different from Naïve Bayes, which
 981 computes corpus statistics and then sets the weights in a single operation — Naïve Bayes
 982 is a **batch learning** algorithm. Algorithm 3 is vague about when this online learning pro-
 983 cedure terminates. We will return to this issue shortly.

984 The perceptron algorithm may seem like a cheap heuristic: Naïve Bayes has a solid
 985 foundation in probability, but the perceptron is just adding and subtracting constants from
 986 the weights every time there is a mistake. Will this really work? In fact, there is some nice
 987 theory for the perceptron, based on the concept of **linear separability**:

988 **Definition 1** (Linear separability). *The dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$ is linearly separable iff
 989 (if and only if) there exists some weight vector $\boldsymbol{\theta}$ and some margin ρ such that for every instance
 990 $(\mathbf{x}^{(i)}, y^{(i)})$, the inner product of $\boldsymbol{\theta}$ and the feature function for the true label, $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)})$, is
 991 at least ρ greater than inner product of $\boldsymbol{\theta}$ and the feature function for every other possible label,
 992 $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y')$.*

$$\exists \boldsymbol{\theta}, \rho > 0 : \forall (\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{D}, \quad \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) \geq \rho + \max_{y' \neq y^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y'). \quad [2.35]$$

993 Linear separability is important because of the following guarantee: if your data is

994 linearly separable, then the perceptron algorithm will find a separator (Novikoff, 1962).¹²
 995 So while the perceptron may seem heuristic, it is guaranteed to succeed, if the learning
 996 problem is easy enough.

997 How useful is this proof? Minsky and Papert (1969) famously proved that the simple
 998 logical function of *exclusive-or* is not separable, and that a perceptron is therefore inca-
 999 pable of learning this function. But this is not just an issue for the perceptron: any linear
 1000 classification algorithm, including Naïve Bayes, will fail on this task. In natural language
 1001 classification problems usually involve high dimensional feature spaces, with thousands
 1002 or millions of features. For these problems, it is very likely that the training data is indeed
 1003 separable. And even if the data is not separable, it is still possible to place an upper bound
 1004 on the number of errors that the perceptron algorithm will make (Freund and Schapire,
 1005 1999).

1006 2.2.2 Averaged perceptron

1007 The perceptron iterates over the data repeatedly — until “tired”, as described in Algo-
 1008 rithm 3. If the data is linearly separable, the perceptron will eventually find a separator,
 1009 and we can stop once all training instances are classified correctly. But if the data is not
 1010 linearly separable, the perceptron can *thrash* between two or more weight settings, never
 1011 converging. In this case, how do we know that we can stop training, and how should
 1012 we choose the final weights? An effective practical solution is to *average* the perceptron
 1013 weights across all iterations.

1014 This procedure is shown in Algorithm 4. The learning algorithm is nearly identical,
 1015 but we also maintain a vector of the sum of the weights, \mathbf{m} . At the end of the learning
 1016 procedure, we divide this sum by the total number of updates t , to compute the average
 1017 weights, $\bar{\theta}$. These average weights are then used for prediction. In the algorithm sketch,
 1018 the average is computed from a running sum, $\mathbf{m} \leftarrow \mathbf{m} + \theta$. However, this is inefficient,
 1019 because it requires $|\theta|$ operations to update the running sum. When $f(\mathbf{x}, y)$ is sparse,
 1020 $|\theta| \gg |f(\mathbf{x}, y)|$ for any individual (\mathbf{x}, y) . This means that computing the running sum will
 1021 be much more expensive than computing of the update to θ itself, which requires only
 1022 $2 \times |f(\mathbf{x}, y)|$ operations. One of the exercises is to sketch a more efficient algorithm for
 1023 computing the averaged weights.

1024 Even if the data is not separable, the averaged weights will eventually converge. One
 1025 possible stopping criterion is to check the difference between the average weight vectors
 1026 after each pass through the data: if the norm of the difference falls below some predefined
 1027 threshold, we can stop training. Another stopping criterion is to hold out some data,
 1028 and to measure the predictive accuracy on this heldout data. When the accuracy on the
 1029 heldout data starts to decrease, the learning algorithm has begun to **overfit** the training

¹²It is also possible to prove an upper bound on the number of training iterations required to find the separator. Proofs like this are part of the field of **statistical learning theory** (Mohri et al., 2012).

Algorithm 4 Averaged perceptron learning algorithm

```

1: procedure AVG-PERCEPTRON( $\mathbf{x}^{(1:N)}, \mathbf{y}^{(1:N)}$ )
2:    $t \leftarrow 0$ 
3:    $\boldsymbol{\theta}^{(0)} \leftarrow 0$ 
4:   repeat
5:      $t \leftarrow t + 1$ 
6:     Select an instance  $i$ 
7:      $\hat{y} \leftarrow \operatorname{argmax}_y \boldsymbol{\theta}^{(t-1)} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y)$ 
8:     if  $\hat{y} \neq y^{(i)}$  then
9:        $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} + \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - \mathbf{f}(\mathbf{x}^{(i)}, \hat{y})$ 
10:    else
11:       $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)}$ 
12:     $\mathbf{m} \leftarrow \mathbf{m} + \boldsymbol{\theta}^{(t)}$ 
13:   until tired
14:    $\bar{\boldsymbol{\theta}} \leftarrow \frac{1}{t} \mathbf{m}$ 
15:   return  $\bar{\boldsymbol{\theta}}$ 

```

1030 set. At this point, it is probably best to stop; this stopping criterion is known as **early**
 1031 **stopping**.

1032 **Generalization** is the ability to make good predictions on instances that are not in
 1033 the training data. Averaging can be proven to improve generalization, by computing an
 1034 upper bound on the generalization error (Freund and Schapire, 1999; Collins, 2002).

1035 **2.3 Loss functions and large-margin classification**

1036 Naïve Bayes chooses the weights $\boldsymbol{\theta}$ by maximizing the joint log-likelihood $\log p(\mathbf{x}^{(1:N)}, \mathbf{y}^{(1:N)})$.
 1037 By convention, optimization problems are generally formulated as minimization of a **loss**
 1038 **function**. The input to a loss function is the vector of weights $\boldsymbol{\theta}$, and the output is a non-
 1039 negative scalar, measuring the performance of the classifier on a training instance. The
 1040 loss $\ell(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)})$ is then a measure of the performance of the weights $\boldsymbol{\theta}$ on the instance
 1041 $(\mathbf{x}^{(i)}, y^{(i)})$. The goal of learning is to minimize the sum of the losses across all instances in
 1042 the training set.

We can trivially reformulate maximum likelihood as a loss function, by defining the

loss function to be the *negative log-likelihood*:

$$\log p(\mathbf{x}^{(1:N)}, y^{(1:N)}; \boldsymbol{\theta}) = \sum_{i=1}^N \log p(\mathbf{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}) \quad [2.36]$$

$$\ell_{\text{NB}}(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)}) = -\log p(\mathbf{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}) \quad [2.37]$$

$$\hat{\boldsymbol{\theta}} = \operatorname{argmin}_{\boldsymbol{\theta}} \sum_{i=1}^N \ell_{\text{NB}}(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)}) \quad [2.38]$$

$$= \operatorname{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^N \log p(\mathbf{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}). \quad [2.39]$$

1043 The problem of minimizing ℓ_{NB} is thus identical to the problem of maximum-likelihood
1044 estimation.

1045 Loss functions provide a general framework for comparing machine learning objectives.
1046 For example, an alternative loss function is the **zero-one loss**,

$$\ell_{0-1}(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)}) = \begin{cases} 0, & y^{(i)} = \operatorname{argmax}_y \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y) \\ 1, & \text{otherwise} \end{cases} \quad [2.40]$$

1047 The zero-one loss is zero if the instance is correctly classified, and one otherwise. The
1048 sum of zero-one losses is proportional to the error rate of the classifier on the training
1049 data. Since a low error rate is often the ultimate goal of classification, this may seem
1050 ideal. But the zero-one loss has several problems. One is that it is **non-convex**,¹³ which
1051 means that there is no guarantee that gradient-based optimization will be effective. A
1052 more serious problem is that the derivatives are useless: the partial derivative with respect
1053 to any parameter is zero everywhere, except at the points where $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y) = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, \hat{y})$
1054 for some \hat{y} . At those points, the loss is discontinuous, and the derivative is undefined.

1055 The perceptron optimizes the following loss function:

$$\ell_{\text{PERCEPTRON}}(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)}) = \max_{y \in \mathcal{Y}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}), \quad [2.41]$$

1056 When $\hat{y} = y^{(i)}$, the loss is zero; otherwise, it increases linearly with the gap between the
1057 score for the predicted label \hat{y} and the score for the true label $y^{(i)}$. Plotting this loss against
1058 the input $\max_{y \in \mathcal{Y}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)})$ gives a hinge shape, motivating the name
1059 **hinge loss**.

¹³A function f is **convex** iff $\alpha f(x_i) + (1-\alpha)f(x_j) \geq f(\alpha x_i + (1-\alpha)x_j)$, for all $\alpha \in [0, 1]$ and for all x_i and x_j on the domain of the function. In words, any weighted average of the output of f applied to any two points is larger than the output of f when applied to the weighted average of the same two points. Convexity implies that any local minimum is also a global minimum, and there are many effective techniques for optimizing convex functions (Boyd and Vandenberghe, 2004). See Appendix B for a brief review.

1060 To see why this is the loss function optimized by the perceptron, take the derivative
 1061 with respect to θ ,

$$\frac{\partial}{\partial \theta} \ell_{\text{PERCEPTRON}}(\theta; \mathbf{x}^{(i)}, y^{(i)}) = \mathbf{f}(\mathbf{x}^{(i)}, \hat{y}) - \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}). \quad [2.42]$$

1062 At each instance perceptron algorithm takes a step of magnitude one in the opposite direction
 1063 of this **gradient**, $\nabla_{\theta} \ell_{\text{PERCEPTRON}} = \frac{\partial}{\partial \theta} \ell_{\text{PERCEPTRON}}(\theta; \mathbf{x}^{(i)}, y^{(i)})$. As we will see in § 2.5,
 1064 this is an example of the optimization algorithm **stochastic gradient descent**, applied to
 1065 the objective in Equation 2.41.

1066 **Breaking ties with subgradient descent** Careful readers will notice the tacit assumption
 1067 that there is a unique \hat{y} that maximizes $\theta \cdot \mathbf{f}(\mathbf{x}^{(i)}, y)$. What if there are two or more labels
 1068 that maximize this function? Consider binary classification: if the maximizer is $y^{(i)}$, then
 1069 the gradient is zero, and so is the perceptron update; if the maximizer is $\hat{y} \neq y^{(i)}$, then the
 1070 update is the difference $\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - \mathbf{f}(\mathbf{x}^{(i)}, \hat{y})$. The underlying issue is that the perceptron
 1071 loss is not **smooth**, because the first derivative has a discontinuity at the hinge point,
 1072 where the score for the true label $y^{(i)}$ is equal to the score for some other label \hat{y} . At this
 1073 point, there is no unique gradient; rather, there is a set of **subgradients**. A vector v is a
 1074 subgradient of the function g at u_0 iff $g(u) - g(u_0) \geq v \cdot (u - u_0)$ for all u . Graphically,
 1075 this defines the set of hyperplanes that include $g(u_0)$ and do not intersect g at any other
 1076 point. As we approach the hinge point from the left, the gradient is $\mathbf{f}(\mathbf{x}, \hat{y}) - \mathbf{f}(\mathbf{x}, y)$; as we
 1077 approach from the right, the gradient is 0. At the hinge point, the subgradients include all
 1078 vectors that are bounded by these two extremes. In subgradient descent, *any* subgradient
 1079 can be used (Bertsekas, 2012). Since both 0 and $\mathbf{f}(\mathbf{x}, \hat{y}) - \mathbf{f}(\mathbf{x}, y)$ are subgradients at the
 1080 hinge point, either one can be used in the perceptron update.

1081 **Perceptron versus Naïve Bayes** The perceptron loss function has some pros and cons
 1082 with respect to the negative log-likelihood loss implied by Naïve Bayes.

- 1083 • Both ℓ_{NB} and $\ell_{\text{PERCEPTRON}}$ are convex, making them relatively easy to optimize. How-
 1084 ever, ℓ_{NB} can be optimized in closed form, while $\ell_{\text{PERCEPTRON}}$ requires iterating over
 1085 the dataset multiple times.
- 1086 • ℓ_{NB} can suffer **infinite** loss on a single example, since the logarithm of zero probabil-
 1087 ity is negative infinity. Naïve Bayes will therefore overemphasize some examples,
 1088 and underemphasize others.
- 1089 • $\ell_{\text{PERCEPTRON}}$ treats all correct answers equally. Even if θ only gives the correct answer
 1090 by a tiny margin, the loss is still zero.

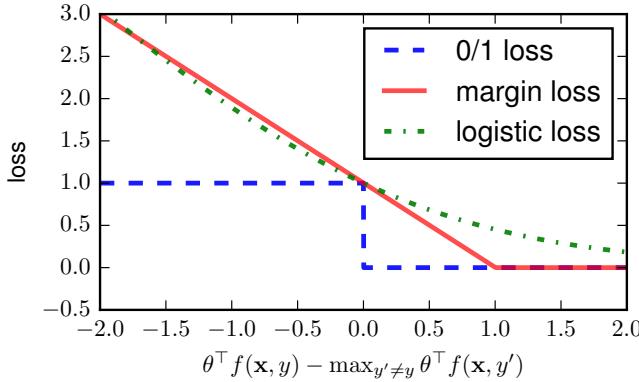


Figure 2.2: Margin, zero-one, and logistic loss functions.

1091 **2.3.1 Large margin classification**

1092 This last comment suggests a potential problem with the perceptron. Suppose a test ex-
 1093 ample is very close to a training example, but not identical. If the classifier only gets the
 1094 correct answer on the training example by a small margin, then it may get the test instance
 1095 wrong. To formalize this intuition, define the **margin** as,

$$\gamma(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)}) = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - \max_{y \neq y^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y). \quad [2.43]$$

The margin represents the difference between the score for the correct label $y^{(i)}$, and the score for the highest-scoring label. The intuition behind **large margin classification** is that it is not enough just to label the training data correctly — the correct label should be separated from other labels by a comfortable margin. This idea can be encoded into a loss function,

$$\ell_{\text{MARGIN}}(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)}) = \begin{cases} 0, & \gamma(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)}) \geq 1, \\ 1 - \gamma(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)}), & \text{otherwise} \end{cases} \quad [2.44]$$

$$= (1 - \gamma(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)}))_+, \quad [2.45]$$

1096 where $(x)_+ = \max(0, x)$. The loss is zero if there is a margin of at least 1 between the
 1097 score for the true label and the best-scoring alternative \hat{y} . This is almost identical to the
 1098 perceptron loss, but the hinge point is shifted to the right, as shown in Figure 2.2. The
 1099 margin loss is a convex upper bound on the zero-one loss.

1100 **2.3.2 Support vector machines**

If a dataset is linearly separable, then there is some hyperplane θ that correctly classifies all training instances with margin ρ (by Definition 1). This margin can be increased to any desired value by multiplying the weights by a constant. Now, for any datapoint $(x^{(i)}, y^{(i)})$, the geometric distance to the separating hyperplane is given by $\frac{\gamma(\theta; x^{(i)}, y^{(i)})}{\|\theta\|_2}$, where the denominator is the norm of the weights, $\|\theta\|_2 = \sqrt{\sum_j \theta_j^2}$. The geometric distance is sometimes called the **geometric margin**, in contrast to the **functional margin** $\gamma(\theta; x^{(i)}, y^{(i)})$. Both are shown in Figure 2.3. The geometric margin is a good measure of the robustness of the separator: if the functional margin is large, but the norm $\|\theta\|_2$ is also large, then a small change in $x^{(i)}$ could cause it to be misclassified. We therefore seek to maximize the minimum geometric margin, subject to the constraint that the functional margin is at least one:

$$\begin{aligned} \max_{\theta} . & \quad \min_i . & & \frac{\gamma(\theta; x^{(i)}, y^{(i)})}{\|\theta\|_2} \\ \text{s.t.} & \quad \gamma(\theta; x^{(i)}, y^{(i)}) \geq 1, \quad \forall i. & & [2.46] \end{aligned}$$

1101 This is a **constrained optimization** problem, where the second line describes constraints
 1102 on the space of possible solutions θ . In this case, the constraint is that the functional
 1103 margin always be at least one, and the objective is that the minimum geometric margin
 1104 be as large as possible.

Any scaling factor on θ will cancel in the numerator and denominator of the geometric margin. This means that if the data is linearly separable at ρ , we can increase this margin to 1 by rescaling θ . We therefore need only minimize the denominator $\|\theta\|_2$, subject to the constraint on the functional margin. The minimizer of $\|\theta\|_2$ is also the minimizer of $\frac{1}{2}\|\theta\|_2^2 = \frac{1}{2}\sum_{j=1}^V \theta_j^2$, which is easier to work with. This gives the optimization problem,

$$\begin{aligned} \min_{\theta} . & \quad \frac{1}{2}\|\theta\|_2^2 \\ \text{s.t.} & \quad \gamma(\theta; x^{(i)}, y^{(i)}) \geq 1, \quad \forall i. & & [2.47] \end{aligned}$$

1105 This optimization problem is a **quadratic program**: the objective is a quadratic function
 1106 of the parameters, and the constraints are all linear inequalities. The resulting classifier
 1107 is better known as the **support vector machine**. The name derives from one of the
 1108 solutions, which is to incorporate the constraints through Lagrange multipliers $\alpha_i \geq 0, i =$
 1109 $1, 2, \dots, N$. The instances for which $\alpha_i > 0$ are the **support vectors**; other instances are
 1110 irrelevant to the classification boundary.

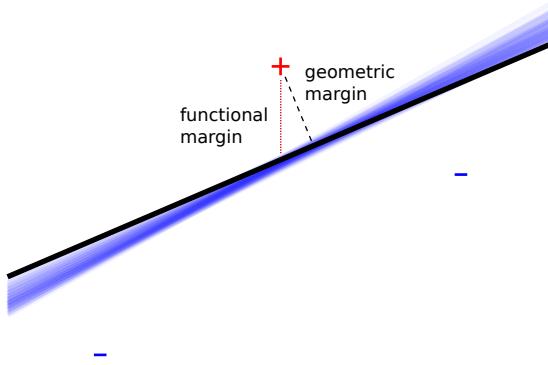


Figure 2.3: Functional and geometric margins for a binary classification problem. All separators that satisfy the margin constraint are shown. The separator with the largest geometric margin is shown in bold.

1111 2.3.3 Slack variables

If a dataset is not linearly separable, then there is no θ that satisfies the margin constraint. To add more flexibility, we introduce a set of **slack variables** $\xi_i \geq 0$. Instead of requiring that the functional margin be greater than or equal to one, we require that it be greater than or equal to $1 - \xi_i$. Ideally there would not be any slack, so the slack variables are penalized in the objective function:

$$\begin{aligned} \min_{\theta, \xi} \quad & \frac{1}{2} \|\theta\|_2^2 + C \sum_{i=1}^N \xi_i \\ \text{s.t.} \quad & \gamma(\theta; \mathbf{x}^{(i)}, y^{(i)}) + \xi_i \geq 1, \quad \forall i \\ & \xi_i \geq 0, \quad \forall i. \end{aligned} \quad [2.48]$$

1112 The hyperparameter C controls the tradeoff between violations of the margin con-
 1113 straint and the preference for a low norm of θ . As $C \rightarrow \infty$, slack is infinitely expensive,
 1114 and there is only a solution if the data is separable. As $C \rightarrow 0$, slack becomes free, and
 1115 there is a trivial solution at $\theta = 0$. Thus, C plays a similar role to the smoothing parame-
 1116 ter in Naïve Bayes (§ 2.1.4), trading off between a close fit to the training data and better
 1117 generalization. Like the smoothing parameter of Naïve Bayes, C must be set by the user,
 1118 typically by maximizing performance on a heldout development set.

1119 To solve the constrained optimization problem defined in Equation 2.48, we can first

1120 solve for the slack variables,

$$\xi_i \geq (1 - \gamma(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)}))_+. \quad [2.49]$$

The inequality is tight, because the slack variables are penalized in the objective, and there is no advantage to increasing them beyond the minimum value (Ratliff et al., 2007; Smith, 2011). The problem can therefore be transformed into the unconstrained optimization,

$$\min_{\boldsymbol{\theta}} \frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2 + \sum_{i=1}^N (1 - \gamma(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)}))_+, \quad [2.50]$$

1121 where each ξ_i has been substituted by the right-hand side of Equation 2.49, and the factor
 1122 of C on the slack variables has been replaced by an equivalent factor of $\lambda = \frac{1}{C}$ on the
 1123 norm of the weights.

1124 Now define the **cost** of a classification error as,¹⁴

$$c(y^{(i)}, \hat{y}) = \begin{cases} 1, & y^{(i)} \neq \hat{y} \\ 0, & \text{otherwise.} \end{cases} \quad [2.51]$$

Equation 2.50 can be rewritten using this cost function,

$$\min_{\boldsymbol{\theta}} \frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2 + \sum_{i=1}^N \left(\max_{y \in \mathcal{Y}} (\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y) + c(y^{(i)}, y)) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) \right)_+. \quad [2.52]$$

1125 This objective maximizes over all $y \in \mathcal{Y}$, in search of labels that are both *strong*, as measured by $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y)$, and *wrong*, as measured by $c(y^{(i)}, y)$. This maximization is known
 1126 as **cost-augmented decoding**, because it augments the maximization objective to favor
 1127 high-cost predictions. If the highest-scoring label is $y = y^{(i)}$, then the margin constraint is
 1128 satisfied, and the loss for this instance is zero. Cost-augmentation is only for learning: it
 1129 is not applied when making predictions on unseen data.

Differentiating Equation 2.52 with respect to the weights gives,

$$\nabla_{\boldsymbol{\theta}} L_{\text{SVM}} = \lambda \boldsymbol{\theta} + \sum_{i=1}^N \mathbf{f}(\mathbf{x}^{(i)}, \hat{y}) - \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) \quad [2.53]$$

$$\hat{y} = \operatorname{argmax}_{y \in \mathcal{Y}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y) + c(y^{(i)}, y), \quad [2.54]$$

1131 where L_{SVM} refers to minimization objective in Equation 2.52. This gradient is very similar
 1132 to the perceptron update. One difference is the additional term $\lambda \boldsymbol{\theta}$, which **regularizes** the

¹⁴We can also define specialized cost functions that heavily penalize especially undesirable errors (Tsacharidis et al., 2004). This idea is revisited in chapter 7.

1133 weights towards 0. The other difference is the cost $c(y^{(i)}, y)$, which is added to $\theta \cdot \mathbf{f}(\mathbf{x}, y)$
 1134 when choosing \hat{y} during training. This term derives from the margin constraint: large
 1135 margin classifiers learn not only from instances that are incorrectly classified, but also
 1136 from instances for which the correct classification decision was not sufficiently confident.

1137 2.4 Logistic regression

1138 Thus far, we have seen two broad classes of learning algorithms. Naïve Bayes is a prob-
 1139 abilistic method, where learning is equivalent to estimating a joint probability distribu-
 1140 tion. The perceptron and support vector machine are discriminative, error-driven algo-
 1141 rithms: the learning objective is closely related to the number of errors on the training
 1142 data. Probabilistic and error-driven approaches each have advantages: probability makes
 1143 it possible to quantify uncertainty about the predicted labels, but the probability model of
 1144 Naïve Bayes makes unrealistic independence assumptions that limit the features that can
 1145 be used.

Logistic regression combines advantages of discriminative and probabilistic classi-
 fiers. Unlike Naïve Bayes, which starts from the **joint probability** $p_{X,Y}$, logistic regression
 defines the desired **conditional probability** $p_{Y|X}$ directly. Think of $\theta \cdot \mathbf{f}(\mathbf{x}, y)$ as a scoring
 function for the compatibility of the base features \mathbf{x} and the label y . To convert this score
 into a probability, we first exponentiate, obtaining $\exp(\theta \cdot \mathbf{f}(\mathbf{x}, y))$, which is guaranteed
 to be non-negative. Next, we normalize, dividing over all possible labels $y' \in \mathcal{Y}$. The
 resulting conditional probability is defined as,

$$p(y | \mathbf{x}; \theta) = \frac{\exp(\theta \cdot \mathbf{f}(\mathbf{x}, y))}{\sum_{y' \in \mathcal{Y}} \exp(\theta \cdot \mathbf{f}(\mathbf{x}, y'))}. \quad [2.55]$$

Given a dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$, the weights θ are estimated by **maximum conditional likelihood**,

$$\log p(\mathbf{y}^{(1:N)} | \mathbf{x}^{(1:N)}; \theta) = \sum_{i=1}^N \log p(y^{(i)} | \mathbf{x}^{(i)}; \theta) \quad [2.56]$$

$$= \sum_{i=1}^N \theta \cdot \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - \log \sum_{y' \in \mathcal{Y}} \exp(\theta \cdot \mathbf{f}(\mathbf{x}^{(i)}, y')). \quad [2.57]$$

1146 The final line is obtained by plugging in Equation 2.55 and taking the logarithm.¹⁵ Inside

¹⁵The log-sum-exp term is a common pattern in machine learning. It is numerically unstable, because it will underflow if the inner product is small, and overflow if the inner product is large. Scientific computing libraries usually contain special functions for computing `logsumexp`, but with some thought, you should be able to see how to create an implementation that is numerically stable.

1147 the sum, we have the (additive inverse of the) **logistic loss**,

$$\ell_{\text{LOGREG}}(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)}) = -\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) + \log \sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y')) \quad [2.58]$$

1148 The logistic loss is shown in Figure 2.2. A key difference from the zero-one and hinge
1149 losses is that logistic loss is never zero. This means that the objective function can always
1150 be improved by assigning higher confidence to the correct label.

1151 2.4.1 Regularization

1152 As with the support vector machine, better generalization can be obtained by penalizing
1153 the norm of $\boldsymbol{\theta}$. This is done by adding a term of $\frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2$ to the minimization objective.
1154 This is called L_2 regularization, because $\|\boldsymbol{\theta}\|_2^2$ is the squared L_2 norm of the vector $\boldsymbol{\theta}$.
1155 Regularization forces the estimator to trade off performance on the training data against
1156 the norm of the weights, and this can help to prevent overfitting. Consider what would
1157 happen to the unregularized weight for a base feature j that is active in only one instance
1158 $\mathbf{x}^{(i)}$: the conditional log-likelihood could always be improved by increasing the weight
1159 for this feature, so that $\boldsymbol{\theta}_{(j,y^{(i)})} \rightarrow \infty$ and $\boldsymbol{\theta}_{(j,\tilde{y} \neq y^{(i)})} \rightarrow -\infty$, where (j, y) is the index of
1160 feature associated with $x_j^{(i)}$ and label y in $\mathbf{f}(\mathbf{x}^{(i)}, y)$.

In § 2.1.4, we saw that smoothing the probabilities of a Naïve Bayes classifier can be justified in a hierarchical probabilistic model, in which the parameters of the classifier are themselves random variables, drawn from a prior distribution. The same justification applies to L_2 regularization. In this case, the prior is a zero-mean Gaussian on each term of $\boldsymbol{\theta}$. The log-likelihood under a zero-mean Gaussian is,

$$\log N(\theta_j; 0, \sigma^2) \propto -\frac{1}{2\sigma^2} \theta_j^2, \quad [2.59]$$

1161 so that the regularization weight λ is equal to the inverse variance of the prior, $\lambda = \frac{1}{\sigma^2}$.

1162 **2.4.2 Gradients**

Logistic loss is minimized by optimization along the gradient. Here is the gradient with respect to the logistic loss on a single example,

$$\ell_{\text{LOGREG}} = -\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) + \log \sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y')) \quad [2.60]$$

$$\frac{\partial \ell}{\partial \boldsymbol{\theta}} = -\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) + \frac{1}{\sum_{y'' \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y''))} \times \sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y')) \times \mathbf{f}(\mathbf{x}^{(i)}, y') \quad [2.61]$$

$$= -\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) + \sum_{y' \in \mathcal{Y}} \frac{\exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y'))}{\sum_{y'' \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y''))} \times \mathbf{f}(\mathbf{x}^{(i)}, y') \quad [2.62]$$

$$= -\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) + \sum_{y' \in \mathcal{Y}} p(y' | \mathbf{x}^{(i)}; \boldsymbol{\theta}) \times \mathbf{f}(\mathbf{x}^{(i)}, y') \quad [2.63]$$

$$= -\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) + E_{Y|X}[\mathbf{f}(\mathbf{x}^{(i)}, y)]. \quad [2.64]$$

1163 The final step employs the definition of a conditional expectation (§ A.5). The gradient of
 1164 the logistic loss is equal to the difference between the expected counts under the current
 1165 model, $E_{Y|X}[\mathbf{f}(\mathbf{x}^{(i)}, y)]$, and the observed feature counts $\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)})$. When these two
 1166 vectors are equal for a single instance, there is nothing more to learn from it; when they
 1167 are equal in sum over the entire dataset, there is nothing more to learn from the dataset as
 1168 a whole. The gradient of the hinge loss is nearly identical, but it involves the features of
 1169 the predicted label under the current model, $\mathbf{f}(\mathbf{x}^{(i)}, \hat{y})$, rather than the expected features
 1170 $E_{Y|X}[\mathbf{f}(\mathbf{x}^{(i)}, y)]$ under the conditional distribution $p(y | \mathbf{x}; \boldsymbol{\theta})$.

The regularizer contributes $\lambda \boldsymbol{\theta}$ to the overall gradient:

$$L_{\text{LOGREG}} = \frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2 - \sum_{i=1}^N \left(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - \log \sum_{y' \in \mathcal{Y}} \exp \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y') \right) \quad [2.65]$$

$$\nabla_{\boldsymbol{\theta}} L_{\text{LOGREG}} = \lambda \boldsymbol{\theta} - \sum_{i=1}^N \left(\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - E_{y|\mathbf{x}}[\mathbf{f}(\mathbf{x}^{(i)}, y)] \right). \quad [2.66]$$

1171 **2.5 Optimization**

1172 Each of the classification algorithms in this chapter can be viewed as an optimization
 1173 problem:

- 1174 • In Naïve Bayes, the objective is the joint likelihood $\log p(\mathbf{x}^{(1:N)}, \mathbf{y}^{(1:N)})$. Maximum
 1175 likelihood estimation yields a closed-form solution for $\boldsymbol{\theta}$.

- 1176 • In the support vector machine, the objective is the regularized margin loss,

$$L_{\text{SVM}} = \frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2 + \sum_{i=1}^N (\max_{y \in \mathcal{Y}} (\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y) + c(y^{(i)}, y)) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}))_+, \quad [2.67]$$

1177 There is no closed-form solution, but the objective is convex. The perceptron algo-
1178 rithm minimizes a similar objective.

- 1179 • In logistic regression, the objective is the regularized negative log-likelihood,

$$L_{\text{LOGREG}} = \frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2 - \sum_{i=1}^N \left(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - \log \sum_{y \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y)) \right) \quad [2.68]$$

1180 Again, there is no closed-form solution, but the objective is convex.

1181 These learning algorithms are distinguished by *what* is being optimized, rather than
1182 *how* the optimal weights are found. This decomposition is an essential feature of con-
1183 temporary machine learning. The domain expert's job is to design an objective function
1184 — or more generally, a **model** of the problem. If the model has certain characteristics,
1185 then generic optimization algorithms can be used to find the solution. In particular, if an
1186 objective function is differentiable, then gradient-based optimization can be employed;
1187 if it is also convex, then gradient-based optimization is guaranteed to find the globally
1188 optimal solution. The support vector machine and logistic regression have both of these
1189 properties, and so are amenable to generic **convex optimization** techniques (Boyd and
1190 Vandenberghe, 2004).

1191 2.5.1 Batch optimization

In **batch optimization**, each update to the weights is based on a computation involving the entire dataset. One such algorithm is **gradient descent**, which iteratively updates the weights,

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta^{(t)} \nabla_{\boldsymbol{\theta}} L, \quad [2.69]$$

1192 where $\nabla_{\boldsymbol{\theta}} L$ is the gradient computed over the entire training set, and $\eta^{(t)}$ is the **step size**
1193 at iteration t . If the objective L is a convex function of $\boldsymbol{\theta}$, then this procedure is guaranteed
1194 to terminate at the global optimum, for appropriate schedule of learning rates, $\eta^{(t)}$.¹⁶

¹⁶Specifically, the learning rate must have the following properties (Bottou et al., 2016):

$$\sum_{t=1}^{\infty} \eta^{(t)} = \infty \quad [2.70]$$

$$\sum_{t=1}^{\infty} (\eta^{(t)})^2 < \infty. \quad [2.71]$$

1195 In practice, gradient descent can be slow to converge, as the gradient can become
 1196 infinitesimally small. Faster convergence can be obtained by second-order Newton opti-
 1197 mization, which incorporates the inverse of the **Hessian matrix**,

$$H_{i,j} = \frac{\partial^2 L}{\partial \theta_i \partial \theta_j} \quad [2.72]$$

1198 The size of the Hessian matrix is quadratic in the number of features. In the bag-of-words
 1199 representation, this is usually too big to store, let alone invert. **Quasi-Network optimiza-**
 1200 **tion** techniques maintain a low-rank approximation to the inverse of the Hessian matrix.
 1201 Such techniques usually converge more quickly than gradient descent, while remaining
 1202 computationally tractable even for large feature sets. A popular quasi-Newton algorithm
 1203 is **L-BFGS** (Liu and Nocedal, 1989), which is implemented in many scientific computing
 1204 environments, such as `scipy` and `Matlab`.

1205 For any gradient-based technique, the user must set the learning rates $\eta^{(t)}$. While con-
 1206 vergence proofs usually employ a decreasing learning rate, in practice, it is common to fix
 1207 $\eta^{(t)}$ to a small constant, like 10^{-3} . The specific constant can be chosen by experimentation,
 1208 although there is research on determining the learning rate automatically (Schaul et al.,
 1209 2013; Wu et al., 2018).

1210 2.5.2 Online optimization

1211 Batch optimization computes the objective on the entire training set before making an up-
 1212 date. This may be inefficient, because at early stages of training, a small number of train-
 1213 ing examples could point the learner in the correct direction. **Online learning** algorithms
 1214 make updates to the weights while iterating through the training data. The theoretical
 1215 basis for this approach is a stochastic approximation to the true objective function,

$$\sum_{i=1}^N \ell(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)}) \approx N \times \ell(\boldsymbol{\theta}; \mathbf{x}^{(j)}, y^{(j)}), \quad (\mathbf{x}^{(j)}, y^{(j)}) \sim \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N, \quad [2.73]$$

1216 where the instance $(\mathbf{x}^{(j)}, y^{(j)})$ is sampled at random from the full dataset.

1217 In **stochastic gradient descent**, the approximate gradient is computed by randomly
 1218 sampling a single instance, and an update is made immediately. This is similar to the
 1219 perceptron algorithm, which also updates the weights one instance at a time. In **mini-**
 1220 **batch** stochastic gradient descent, the gradient is computed over a small set of instances.
 1221 A typical approach is to set the minibatch size so that the entire batch fits in memory on a
 1222 graphics processing unit (GPU; Neubig et al., 2017). It is then possible to speed up learn-
 1223 ing by parallelizing the computation of the gradient over each instance in the minibatch.

These properties can be obtained by the learning rate schedule $\eta^{(t)} = \eta^{(0)} t^{-\alpha}$ for $\alpha \in [1, 2]$.

Algorithm 5 Generalized gradient descent. The function BATCHER partitions the training set into B batches such that each instance appears in exactly one batch. In gradient descent, $B = 1$; in stochastic gradient descent, $B = N$; in minibatch stochastic gradient descent, $1 < B < N$.

```

1: procedure GRADIENT-DESCENT( $\mathbf{x}^{(1:N)}, \mathbf{y}^{(1:N)}, L, \eta^{(1:\infty)}$ , BATCHER,  $T_{\max}$ )
2:    $\boldsymbol{\theta} \leftarrow \mathbf{0}$ 
3:    $t \leftarrow 0$ 
4:   repeat
5:      $(\mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \dots, \mathbf{b}^{(B)}) \leftarrow \text{BATCHER}(N)$ 
6:     for  $n \in \{1, 2, \dots, B\}$  do
7:        $t \leftarrow t + 1$ 
8:        $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \eta^{(t)} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t-1)}; \mathbf{x}^{(b_1^{(n)}, b_2^{(n)}, \dots)}, \mathbf{y}^{(b_1^{(n)}, b_2^{(n)}, \dots)})$ 
9:       if Converged( $\boldsymbol{\theta}^{(1, 2, \dots, t)}$ ) then
10:        return  $\boldsymbol{\theta}^{(t)}$ 
11:   until  $t \geq T_{\max}$ 
12:   return  $\boldsymbol{\theta}^{(t)}$ 

```

1224 Algorithm 5 offers a generalized view of gradient descent. In standard gradient de-
 1225 scendent, the batcher returns a single batch with all the instances. In stochastic gradient de-
 1226 scendent, it returns N batches with one instance each. In mini-batch settings, the batcher
 1227 returns B minibatches, $1 < B < N$.

There are many other techniques for online learning, and the field is currently quite active (Bottou et al., 2016). Some algorithms use an adaptive step size, which can be different for every feature (Duchi et al., 2011). Features that occur frequently are likely to be updated frequently, so it is best to use a small step size; rare features will be updated infrequently, so it is better to take larger steps. The **AdaGrad** (adaptive gradient) algorithm achieves this behavior by storing the sum of the squares of the gradients for each feature, and rescaling the learning rate by its inverse:

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}; \mathbf{x}^{(i)}, y^{(i)}) \quad [2.74]$$

$$\theta_j^{(t+1)} \leftarrow \theta_j^{(t)} - \frac{\eta^{(t)}}{\sqrt{\sum_{t'=1}^t g_{t,j}^2}} g_{t,j}, \quad [2.75]$$

1228 where j iterates over features in $\mathbf{f}(\mathbf{x}, y)$.

1229 In most cases, the number of active features for any instance is much smaller than the
 1230 number of weights. If so, the computation cost of online optimization will be dominated
 1231 by the update from the regularization term, $\lambda \boldsymbol{\theta}$. The solution is to be “lazy”, updating
 1232 each θ_j only as it is used. To implement lazy updating, store an additional parameter τ_j ,
 1233 which is the iteration at which θ_j was last updated. If θ_j is needed at time t , the $t - \tau$

1234 regularization updates can be performed all at once. This strategy is described in detail
 1235 by Kummerfeld et al. (2015).

1236 2.6 *Additional topics in classification

1237 Throughout this text, advanced topics will be marked with an asterisk.

1238 2.6.1 Feature selection by regularization

1239 In logistic regression and large-margin classification, generalization can be improved by
 1240 regularizing the weights towards 0, using the L_2 norm. But rather than encouraging
 1241 weights to be small, it might be better for the model to be **sparse**: it should assign weights
 1242 of exactly zero to most features, and only assign non-zero weights to features that are
 1243 clearly necessary. This idea can be formalized by the L_0 norm, $L_0 = \|\theta\|_0 = \sum_j \delta(\theta_j \neq 0)$,
 1244 which applies a constant penalty for each non-zero weight. This norm can be thought
 1245 of as a form of **feature selection**: optimizing the L_0 -regularized conditional likelihood is
 1246 equivalent to trading off the log-likelihood against the number of active features. Reduc-
 1247 ing the number of active features is desirable because the resulting model will be fast,
 1248 low-memory, and should generalize well, since irrelevant features will be pruned away.
 1249 Unfortunately, the L_0 norm is non-convex and non-differentiable. Optimization under L_0
 1250 regularization is **NP-hard**, meaning that it can be solved efficiently only if P=NP (Ge et al.,
 1251 2011).

1252 A useful alternative is the L_1 norm, which is equal to the sum of the absolute values
 1253 of the weights, $\|\theta\|_1 = \sum_j |\theta_j|$. The L_1 norm is convex, and can be used as an approxima-
 1254 tion to L_0 (Tibshirani, 1996). Conveniently, the L_1 norm also performs feature selection,
 1255 by driving many of the coefficients to zero; it is therefore known as a **sparsity inducing**
 1256 **regularizer**. The L_1 norm does not have a gradient at $\theta_j = 0$, so we must instead optimize
 1257 the L_1 -regularized objective using **subgradient** methods. The associated stochastic sub-
 1258 gradient descent algorithms are only somewhat more complex than conventional SGD;
 1259 Sra et al. (2012) survey approaches for estimation under L_1 and other regularizers.

1260 Gao et al. (2007) compare L_1 and L_2 regularization on a suite of NLP problems, finding
 1261 that L_1 regularization generally gives similar accuracy to L_2 regularization, but that L_1
 1262 regularization produces models that are between ten and fifty times smaller, because more
 1263 than 90% of the feature weights are set to zero.

1264 2.6.2 Other views of logistic regression

In binary classification, we can dispense with the feature function, and choose y based on
 the inner product of $\theta \cdot x$. The conditional probability $p_{Y|X}$ is obtained by passing this

inner product through a **logistic function**,

$$\sigma(a) \triangleq \frac{\exp(a)}{1 + \exp(a)} = (1 + \exp(-a))^{-1} \quad [2.76]$$

$$p(y | \mathbf{x}; \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta} \cdot \mathbf{x}). \quad [2.77]$$

1265 This is the origin of the name **logistic regression**. Logistic regression can be viewed as
 1266 part of a larger family of **generalized linear models** (GLMs), in which various other “link
 1267 functions” convert between the inner product $\boldsymbol{\theta} \cdot \mathbf{x}$ and the parameter of a conditional
 1268 probability distribution.

1269 In the early NLP literature, logistic regression is frequently called **maximum entropy**
 1270 classification (Berger et al., 1996). This name refers to an alternative formulation, in
 1271 which the goal is to find the maximum entropy probability function that satisfies **moment-**
 1272 **matching** constraints. These constraints specify that the empirical counts of each feature
 1273 should match the expected counts under the induced probability distribution $p_{Y|X;\boldsymbol{\theta}}$.

$$\sum_{i=1}^N f_j(\mathbf{x}^{(i)}, y^{(i)}) = \sum_{i=1}^N \sum_{y \in \mathcal{Y}} p(y | \mathbf{x}^{(i)}; \boldsymbol{\theta}) f_j(\mathbf{x}^{(i)}, y), \quad \forall j \quad [2.78]$$

1274 The moment-matching constraint is satisfied exactly when the derivative of the condi-
 1275 tional log-likelihood function (Equation 2.64) is equal to zero. However, the constraint
 1276 can be met by many values of $\boldsymbol{\theta}$, so which should we choose?

1277 The **entropy** of the conditional probability distribution $p_{Y|X}$ is,

$$H(p_{Y|X}) = - \sum_{\mathbf{x} \in \mathcal{X}} p_X(\mathbf{x}) \sum_{y \in \mathcal{Y}} p_{Y|X}(y | \mathbf{x}) \log p_{Y|X}(y | \mathbf{x}), \quad [2.79]$$

1278 where \mathcal{X} is the set of all possible feature vectors, and $p_X(\mathbf{x})$ is the probability of observing
 1279 the base features \mathbf{x} . The distribution p_X is unknown, but it can be estimated by summing
 1280 over all the instances in the training set,

$$\tilde{H}(p_{Y|X}) = - \frac{1}{N} \sum_{i=1}^N \sum_{y \in \mathcal{Y}} p_{Y|X}(y | \mathbf{x}^{(i)}) \log p_{Y|X}(y | \mathbf{x}^{(i)}). \quad [2.80]$$

1281 If the entropy is large, the likelihood function is smooth across possible values of y ;
 1282 if it is small, the likelihood function is sharply peaked at some preferred value; in the
 1283 limiting case, the entropy is zero if $p(y | x) = 1$ for some y . The maximum-entropy cri-
 1284 terion chooses to make the weakest commitments possible, while satisfying the moment-
 1285 matching constraints from Equation 2.78. The solution to this constrained optimization
 1286 problem is identical to the maximum conditional likelihood (logistic-loss) formulation
 1287 that was presented in § 2.4.

1288 2.7 Summary of learning algorithms

1289 It is natural to ask which learning algorithm is best, but the answer depends on what
 1290 characteristics are important to the problem you are trying to solve.

1291 **Naïve Bayes** *Pros:* easy to implement; estimation is fast, requiring only a single pass over
 1292 the data; assigns probabilities to predicted labels; controls overfitting with smoothing
 1293 parameter. *Cons:* often has poor accuracy, especially with correlated features.

1294 **Perceptron** *Pros:* easy to implement; online; error-driven learning means that accuracy
 1295 is typically high, especially after averaging. *Cons:* not probabilistic; hard to know
 1296 when to stop learning; lack of margin can lead to overfitting.

1297 **Support vector machine** *Pros:* optimizes an error-based metric, usually resulting in high
 1298 accuracy; overfitting is controlled by a regularization parameter. *Cons:* not proba-
 1299 bilistic.

1300 **Logistic regression** *Pros:* error-driven and probabilistic; overfitting is controlled by a reg-
 1301 ularization parameter. *Cons:* batch learning requires black-box optimization; logistic
 1302 loss can “overtrain” on correctly labeled examples.

1303 One of the main distinctions is whether the learning algorithm offers a probability
 1304 over labels. This is useful in modular architectures, where the output of one classifier
 1305 is the input for some other system. In cases where probability is not necessary, the sup-
 1306 port vector machine is usually the right choice, since it is no more difficult to implement
 1307 than the perceptron, and is often more accurate. When probability is necessary, logistic
 1308 regression is usually more accurate than Naïve Bayes.

1309 Additional resources

1310 For more on classification, you can consult a textbook on machine learning (e.g., Mur-
 1311 phy, 2012), although the notation will differ slightly from what is typical in natural lan-
 1312 guage processing. Probabilistic methods are surveyed by Hastie et al. (2009), and Mohri
 1313 et al. (2012) emphasize theoretical considerations. Online learning is a rapidly moving
 1314 subfield of machine learning, and Bottou et al. (2016) describes progress through 2016.
 1315 Kummerfeld et al. (2015) empirically review several optimization algorithms for large-
 1316 margin learning. The python toolkit `scikit-learn` includes implementations of all of
 1317 the algorithms described in this chapter (Pedregosa et al., 2011).

1318 **Exercises**

- 1319 1. Let \mathbf{x} be a bag-of-words vector such that $\sum_{j=1}^V x_j = 1$. Verify that the multinomial
1320 probability $p_{\text{mult}}(\mathbf{x}; \phi)$, as defined in Equation 2.12, is identical to the probability of
1321 the same document under a categorical distribution, $p_{\text{cat}}(\mathbf{w}; \phi)$.
- 1322 2. Derive the maximum-likelihood estimate for the parameter μ in Naïve Bayes.
- 1323 3. As noted in the discussion of averaged perceptron in § 2.2.2, the computation of the
1324 running sum $\mathbf{m} \leftarrow \mathbf{m} + \boldsymbol{\theta}$ is unnecessarily expensive, requiring $K \times V$ operations.
1325 Give an alternative way to compute the averaged weights $\bar{\boldsymbol{\theta}}$, with complexity that is
1326 independent of V and linear in the sum of feature sizes $\sum_{i=1}^N |\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)})|$.
- 1327 4. Consider a dataset that is comprised of two identical instances $\mathbf{x}^{(1)} = \mathbf{x}^{(2)}$ with
1328 distinct labels $y^{(1)} \neq y^{(2)}$. Assume all features are binary $x_j \in \{0, 1\}$ for all j .

1329 Now suppose that the averaged perceptron always chooses $i = 1$ when t is even,
1330 and $i = 2$ when t is odd, and that it will terminate under the following condition:

$$\epsilon \geq \max_j \left| \frac{1}{t} \sum_t \theta_j^{(t)} - \frac{1}{t-1} \sum_t \theta_j^{(t-1)} \right|. \quad [2.81]$$

1331 In words, the algorithm stops when the largest change in the averaged weights is
1332 less than or equal to ϵ . Compute the number of iterations before the averaged per-
1333 ceptron terminates.

- 1334 5. Suppose you have two labeled datasets D_1 and D_2 , with the same features and la-
1335 bels.
 - 1336 • Let $\boldsymbol{\theta}^{(1)}$ be the unregularized logistic regression (LR) coefficients from training
1337 on dataset D_1 .
 - 1338 • Let $\boldsymbol{\theta}^{(2)}$ be the unregularized LR coefficients (same model) from training on
1339 dataset D_2 .
 - 1340 • Let $\boldsymbol{\theta}^*$ be the unregularized LR coefficients from training on the combined
1341 dataset $D_1 \cup D_2$.

Under these conditions, prove that for any feature j ,

$$\begin{aligned} \theta_j^* &\geq \min(\theta_j^{(1)}, \theta_j^{(2)}) \\ \theta_j^* &\leq \max(\theta_j^{(1)}, \theta_j^{(2)}). \end{aligned}$$

1342

1343 **Chapter 3**

1344 **Nonlinear classification**

1345 Linear classification may seem like all we need for natural language processing. The bag-
1346 of-words representation is inherently high dimensional, and the number of features is
1347 often larger than the number of training instances. This means that it is usually possible
1348 to find a linear classifier that perfectly fits the training data. Moving to nonlinear classifi-
1349 cation may therefore only increase the risk of overfitting. For many tasks, **lexical features**
1350 (words) are meaningful in isolation, and can offer independent evidence about the in-
1351 stance label — unlike computer vision, where individual pixels are rarely informative,
1352 and must be evaluated holistically to make sense of an image. For these reasons, natu-
1353 ral language processing has historically focused on linear classification to a greater extent
1354 than other machine learning application domains.

1355 But in recent years, nonlinear classifiers have swept through natural language pro-
1356 cessing, and are now the default approach for many tasks (Manning, 2016). There are at
1357 least three reasons for this change.

- 1358 • There have been rapid advances in **deep learning**, a family of nonlinear meth-
1359 ods that learn complex functions of the input through multiple layers of computa-
1360 tion (Goodfellow et al., 2016).
- 1361 • Deep learning facilitates the incorporation of **word embeddings**, which are dense
1362 vector representations of words. Word embeddings can be learned from large amounts
1363 of unlabeled data, and enable generalization to words that do not appear in the an-
1364notated training data (word embeddings are discussed in detail in chapter 14).
- 1365 • A third reason for the rise of deep nonlinear learning algorithms is hardware. Many
1366 deep learning models can be implemented efficiently on graphics processing units
1367 (GPUs), offering substantial performance improvements over CPU-based comput-
1368 ing.

1369 This chapter focuses on **neural networks**, which are the dominant approach for non-

1370 linear classification in natural language processing today.¹ Historically, a few other non-
 1371 linear learning methods have been applied to language data:

- 1372 • **Kernel methods** are generalizations of the **nearest-neighbor** classification rule, which
 1373 classifies each instance by the label of the most similar example in the training
 1374 set (Hastie et al., 2009). The application of the **kernel support vector machine** to
 1375 information extraction is described in chapter 17.
- 1376 • **Decision trees** classify instances by checking a set of conditions. Scaling decision
 1377 trees to bag-of-words inputs is difficult, but decision trees have been successful in
 1378 problems such as coreference resolution (chapter 15), where more compact feature
 1379 sets can be constructed (Soon et al., 2001).
- 1380 • **Boosting** and related **ensemble methods** work by combining the predictions of sev-
 1381 eral “weak” classifiers, each of which may consider only a small subset of features.
 1382 Boosting has been successfully applied to text classification (Schapire and Singer,
 1383 2000) and syntactic analysis (Abney et al., 1999), and remains one of the most suc-
 1384 cessful methods on machine learning competition sites such as Kaggle (Chen and
 1385 Guestrin, 2016).

1386 3.1 Feedforward neural networks

1387 Consider the problem of building a classifier for movie reviews. The goal is to predict
 1388 a label $y \in \{\text{GOOD}, \text{BAD}, \text{OKAY}\}$ from a representation of the text of each document, x .
 1389 But what makes a good movie? The story, acting, cinematography, soundtrack, and so
 1390 on. Now suppose the training set contains labels for each of these additional features,
 1391 $z = [z_1, z_2, \dots, z_{K_z}]^\top$. With such information, we could build a two-step classifier:

- 1392 1. **Use the text x to predict the features z .** Specifically, train a logistic regression clas-
 1393 sifier to compute $p(z_k | x)$, for each $k \in \{1, 2, \dots, K_z\}$.
- 1394 2. **Use the features z to predict the label y .** Again, train a logistic regression classifier
 1395 to compute $p(y | z)$. On test data, z is unknown, so we use the probabilities $p(z | x)$
 1396 from the first layer as the features.

1397 This setup is shown in Figure 3.1, which describes the proposed classifier in a **compu-**
 1398 **tation graph**: the text features x are connected to the middle layer z , which in turn is
 1399 connected to the label y .

1400 Since each $z_k \in \{0, 1\}$, we can treat $p(z_k | x)$ as a binary classification problem, using
 1401 binary logistic regression:

$$\Pr(z_k = 1 | x; \Theta^{(x \rightarrow z)}) = \sigma(\boldsymbol{\theta}_k^{(x \rightarrow z)} \cdot x) = (1 + \exp(-\boldsymbol{\theta}_k^{(x \rightarrow z)} \cdot x))^{-1}, \quad [3.1]$$

¹I will use “deep learning” and “neural networks” interchangeably.

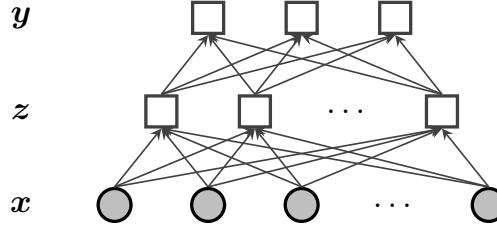


Figure 3.1: A feedforward neural network. Shaded circles indicate observed features, usually words; squares indicate nodes in the computation graph, which are computed from the information carried over the incoming arrows.

1402 where $\sigma(\cdot)$ is the **sigmoid** function (shown in Figure 3.2), and the matrix $\Theta^{(x \rightarrow z)} \in \mathbb{R}^{K_z \times V}$
 1403 is constructed by stacking the weight vectors for each z_k ,

$$\Theta^{(x \rightarrow z)} = [\theta_1^{(x \rightarrow z)}, \theta_2^{(x \rightarrow z)}, \dots, \theta_{K_z}^{(x \rightarrow z)}]^\top. \quad [3.2]$$

1404 We will assume that x contains a term with a constant value of 1, so that a corresponding
 1405 offset parameter is included in each $\theta_k^{(x \rightarrow z)}$.

1406 The output layer is computed by the multi-class logistic regression probability,

$$\Pr(y = j \mid z; \Theta^{(z \rightarrow y)}, b) = \frac{\exp(\theta_j^{(z \rightarrow y)} \cdot z + b_j)}{\sum_{j' \in \mathcal{Y}} \exp(\theta_{j'}^{(z \rightarrow y)} \cdot z + b_{j'})}, \quad [3.3]$$

1407 where b_j is an offset for label j , and the output weight matrix $\Theta^{(z \rightarrow y)} \in \mathbb{R}^{K_y \times K_z}$ is again
 1408 constructed by concatenation,

$$\Theta^{(z \rightarrow y)} = [\theta_1^{(z \rightarrow y)}, \theta_2^{(z \rightarrow y)}, \dots, \theta_{K_y}^{(z \rightarrow y)}]^\top. \quad [3.4]$$

1409 The vector of probabilities over each possible value of y is denoted,

$$p(y \mid z; \Theta^{(z \rightarrow y)}, b) = \text{SoftMax}(\Theta^{(z \rightarrow y)} z + b), \quad [3.5]$$

1410 where element j in the output of the **SoftMax** function is computed as in Equation 3.3.

We have now defined a multilayer classifier, which can be summarized as,

$$p(z \mid x; \Theta^{(x \rightarrow z)}) = \sigma(\Theta^{(x \rightarrow z)} x) \quad [3.6]$$

$$p(y \mid z; \Theta^{(z \rightarrow y)}, b) = \text{SoftMax}(\Theta^{(z \rightarrow y)} z + b), \quad [3.7]$$

1411 where $\sigma(\cdot)$ is now applied **elementwise** to the vector of inner products,

$$\sigma(\Theta^{(x \rightarrow z)} x) = [\sigma(\theta_1^{(x \rightarrow z)} \cdot x), \sigma(\theta_2^{(x \rightarrow z)} \cdot x), \dots, \sigma(\theta_{K_z}^{(x \rightarrow z)} \cdot x)]^\top. \quad [3.8]$$

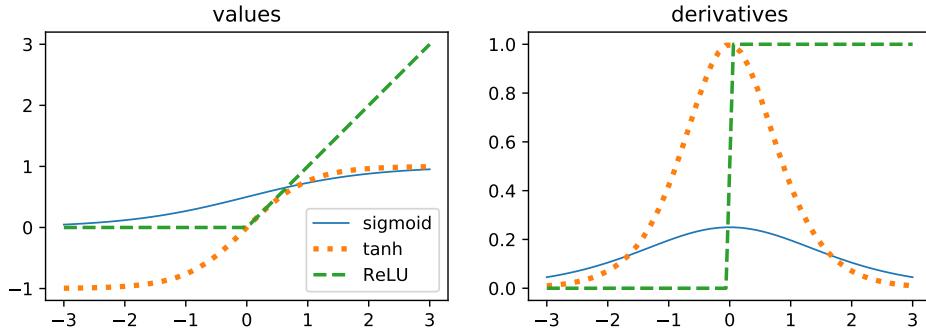


Figure 3.2: The sigmoid, tanh, and ReLU activation functions

Now suppose that the hidden features z are never observed, even in the training data. We can still construct the architecture in Figure 3.1. Instead of predicting y from a discrete vector of predicted values z , we use the probabilities $\sigma(\theta_k \cdot x)$. The resulting classifier is barely changed:

$$z = \sigma(\Theta^{(x \rightarrow z)} x) \quad [3.9]$$

$$p(y | x; \Theta^{(z \rightarrow y)}, b) = \text{SoftMax}(\Theta^{(z \rightarrow y)} z + b). \quad [3.10]$$

1412 This defines a classification model that predicts the label $y \in \mathcal{Y}$ from the base features x ,
1413 through a “hidden layer” z . This is a **feedforward neural network**.²

1414 3.2 Designing neural networks

1415 This feedforward neural network can be generalized in a number of ways.

1416 3.2.1 Activation functions

1417 If the hidden layer is viewed as a set of latent features, then the sigmoid function repre-
1418 sents the extent to which each of these features is “activated” by a given input. However,
1419 the hidden layer can be regarded more generally as a nonlinear transformation of the in-
1420 put. This opens the door to many other activation functions, some of which are shown in
1421 Figure 3.2. At the moment, the choice of activation functions is more art than science, but
1422 a few points can be made about the most popular varieties:

- 1423 • The range of the sigmoid function is $(0, 1)$. The bounded range ensures that a cas-
1424 cade of sigmoid functions will not “blow up” to a huge output, and this is impor-

²The architecture is sometimes called a **multilayer perceptron**, but this is misleading, because each layer is not a perceptron as defined in Algorithm 3.

tant for deep networks with several hidden layers. The derivative of the sigmoid is $\frac{\partial}{\partial a} \sigma(a) = \sigma(a)(1 - \sigma(a))$. This derivative becomes small at the extremes, which can make learning slow; this is called the **vanishing gradient** problem.

- The range of the **tanh activation function** is $(-1, 1)$: like the sigmoid, the range is bounded, but unlike the sigmoid, it includes negative values. The derivative is $\frac{\partial}{\partial a} \tanh(a) = 1 - \tanh(a)^2$, which is steeper than the logistic function near the origin (LeCun et al., 1998). The tanh function can also suffer from vanishing gradients at extreme values.
- The **rectified linear unit (ReLU)** is zero for negative inputs, and linear for positive inputs (Glorot et al., 2011),

$$\text{ReLU}(a) = \begin{cases} a, & a \geq 0 \\ 0, & \text{otherwise.} \end{cases} \quad [3.11]$$

The derivative is a step function, which is 1 if the input is positive, and zero otherwise. Once the activation is zero, the gradient is also zero. This can lead to the problem of **dead neurons**, where some ReLU nodes are zero for all inputs, throughout learning. A solution is the **leaky ReLU**, which has a small positive slope for negative inputs (Maas et al., 2013),

$$\text{Leaky-ReLU}(a) = \begin{cases} a, & a \geq 0 \\ .0001a, & \text{otherwise.} \end{cases} \quad [3.12]$$

Sigmoid and tanh are sometimes described as **squashing functions**, because they squash an unbounded input into a bounded range. Glorot and Bengio (2010) recommend against the use of the sigmoid activation in deep networks, because its mean value of $\frac{1}{2}$ can cause the next layer of the network to be saturated, with very small gradients on their own parameters. Several other activation functions are reviewed by Goodfellow et al. (2016), who recommend ReLU as the “default option.”

3.2.2 Network structure

Deep networks stack up several hidden layers, with each $\mathbf{z}^{(d)}$ acting as the input to the next layer, $\mathbf{z}^{(d+1)}$. As the total number of nodes in the network increases, so does its capacity to learn complex functions of the input. For a fixed number of nodes, an architectural decision is whether to emphasize width (large K_z at each layer) or depth (many layers). At present, this tradeoff is not well understood.³

³With even a single hidden layer, a neural network can approximate any continuous function on a closed and bounded subset of \mathbb{R}^N to an arbitrarily small non-zero error; see section 6.4.1 of Goodfellow et al. (2016) for a survey of these theoretical results. However, depending on the function to be approximated, the width

1452 It is also possible to “short circuit” a hidden layer, by propagating information directly
 1453 from the input to the next higher level of the network. This is the idea behind **residual net-**
 1454 **works**, which propagate information directly from the input to the subsequent layer (He
 1455 et al., 2016),

$$z = f(\Theta^{(x \rightarrow z)} \mathbf{x}) + \mathbf{x}, \quad [3.13]$$

where f is any nonlinearity, such as sigmoid or ReLU. A more complex architecture is the **highway network** (Srivastava et al., 2015; Kim et al., 2016), in which an addition **gate** controls an interpolation between $f(\Theta^{(x \rightarrow z)} \mathbf{x})$ and \mathbf{x} :

$$t = \sigma(\Theta^{(t)} \mathbf{x} + \mathbf{b}^{(t)}) \quad [3.14]$$

$$z = t \odot f(\Theta^{(x \rightarrow z)} \mathbf{x}) + (1 - t) \odot \mathbf{x}, \quad [3.15]$$

1456 where \odot refers to an elementwise vector product, and $\mathbf{1}$ is a column vector of ones. The
 1457 sigmoid function is applied elementwise to its input; recall that the output of this function
 1458 is restricted to the range $[0, 1]$. Gating is also used in the **long short-term memory (LSTM)**,
 1459 which is discussed in chapter 6. Residual and highway connections address a problem
 1460 with deep architectures: repeated application of a nonlinear activation function can make
 1461 it difficult to learn the parameters of the lower levels of the network, which are too distant
 1462 from the supervision signal.

1463 3.2.3 Outputs and loss functions

In the multi-class classification example, a softmax output produces probabilities over each possible label. This aligns with a negative **conditional log-likelihood**,

$$-\mathcal{L} = -\sum_{i=1}^N \log p(y^{(i)} | \mathbf{x}^{(i)}; \Theta). \quad [3.16]$$

1464 where $\Theta = \{\Theta^{(x \rightarrow z)}, \Theta^{(z \rightarrow y)}, \mathbf{b}\}$ is the entire set of parameters.

This loss can be written alternatively as follows:

$$\tilde{y}_j \triangleq \Pr(y = j | \mathbf{x}^{(i)}; \Theta) \quad [3.17]$$

$$-\mathcal{L} = -\sum_{i=1}^N e_{y^{(i)}} \cdot \log \tilde{y} \quad [3.18]$$

1465 where $e_{y^{(i)}}$ is a **one-hot vector** of zeros with a value of 1 at position $y^{(i)}$. The inner product
 1466 between $e_{y^{(i)}}$ and $\log \tilde{y}$ is also called the multinomial **cross-entropy**, and this terminology
 1467 is preferred in many neural networks papers and software packages.

of the hidden layer may need to be arbitrarily large. Furthermore, the fact that a network has the capacity to approximate any given function does not say anything about whether it is possible to *learn* the function using gradient-based optimization.

It is also possible to train neural networks from other objectives, such as a margin loss. In this case, it is not necessary to use softmax at the output layer: an affine transformation of the hidden layer is enough:

$$\Psi(y; \mathbf{x}^{(i)}, \Theta) = \theta_y^{(z \rightarrow y)} \cdot \mathbf{z} + b_y \quad [3.19]$$

$$\ell_{\text{MARGIN}}(\Theta; \mathbf{x}^{(i)}, y^{(i)}) = \max_{y \neq y^{(i)}} \left(1 + \Psi(y; \mathbf{x}^{(i)}, \Theta) - \Psi(y^{(i)}; \mathbf{x}^{(i)}, \Theta) \right)_+ \quad [3.20]$$

- 1468 In regression problems, the output is a scalar or vector (see § 4.1.2). For these problems, a
 1469 typical loss function is the squared error $(y - \hat{y})^2$ or squared norm $\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2$.

1470 3.2.4 Inputs and lookup layers

1471 In text classification, the input layer \mathbf{x} can refer to a bag-of-words vector, where x_j is
 1472 the count of word j . The input to the hidden unit z_k is then $\sum_{j=1}^V \theta_{j,k}^{(x \rightarrow z)} x_j$, and word j is
 1473 represented by the vector $\theta_j^{(x \rightarrow z)}$. This vector is sometimes described as the **embedding** of
 1474 word j , and can be learned from unlabeled data, using techniques discussed in chapter 14.
 1475 The columns of $\Theta^{(x \rightarrow z)}$ are each K_z -dimensional word embeddings.

1476 Chapter 2 presented an alternative view of text documents, as a sequence of word
 1477 tokens, w_1, w_2, \dots, w_M . In a neural network, each word token w_m is represented with
 1478 a one-hot vector, $e_{w_m} \in \mathbb{R}^V$. The matrix-vector product $\Theta^{(x \rightarrow z)} e_{w_m}$ returns the embed-
 1479 ding of word w_m . The complete document can be represented by horizontally concatenating
 1480 these one-hot vectors, $\mathbf{W} = [e_{w_1}, e_{w_2}, \dots, e_{w_M}]$, and the bag-of-words representation can
 1481 be recovered from the matrix-vector product $\mathbf{W} \mathbf{1}$, which simply sums each row over the
 1482 tokens $m = \{1, 2, \dots, M\}$. The matrix product $\Theta^{(x \rightarrow z)} \mathbf{W}$ contains the horizontally con-
 1483 catenated embeddings of each word in the document, which will be useful as the starting
 1484 point for **convolutional neural networks** (see § 3.4). This is sometimes called a **lookup**
 1485 **layer**, because the first step is to lookup the embeddings for each word in the input text.

1486 3.3 Learning neural networks

The feedforward network in Figure 3.1 can now be written in a more general form,

$$\mathbf{z} \leftarrow f(\Theta^{(x \rightarrow z)} \mathbf{x}^{(i)}) \quad [3.21]$$

$$\tilde{\mathbf{y}} \leftarrow \text{SoftMax} \left(\Theta^{(z \rightarrow y)} \mathbf{z} + \mathbf{b} \right) \quad [3.22]$$

$$\ell^{(i)} \leftarrow -e_{y^{(i)}} \cdot \log \tilde{y}, \quad [3.23]$$

- 1487 where f is an elementwise activation function, such as σ or ReLU.

Let us now consider how to estimate the parameters $\Theta^{(x \rightarrow z)}$, $\Theta^{(z \rightarrow y)}$ and \mathbf{b} , using online gradient-based optimization. The simplest such algorithm is stochastic gradient descent (Algorithm 5). The relevant updates are,

$$\mathbf{b} \leftarrow \mathbf{b} - \eta^{(t)} \nabla_{\mathbf{b}} \ell^{(i)} \quad [3.24]$$

$$\boldsymbol{\theta}_k^{(z \rightarrow y)} \leftarrow \boldsymbol{\theta}_k^{(z \rightarrow y)} - \eta^{(t)} \nabla_{\boldsymbol{\theta}_k^{(z \rightarrow y)}} \ell^{(i)} \quad [3.25]$$

$$\boldsymbol{\theta}_k^{(x \rightarrow z)} \leftarrow \boldsymbol{\theta}_k^{(x \rightarrow z)} - \eta^{(t)} \nabla_{\boldsymbol{\theta}_k^{(x \rightarrow z)}} \ell^{(i)}, \quad [3.26]$$

where $\eta^{(t)}$ is the learning rate on iteration t , $\ell^{(i)}$ is the loss at instance (or minibatch) i , and $\boldsymbol{\theta}_k^{(x \rightarrow z)}$ is column k of the matrix $\Theta^{(x \rightarrow z)}$, and $\boldsymbol{\theta}_k^{(z \rightarrow y)}$ is column k of $\Theta^{(z \rightarrow y)}$.

The gradients of the negative log-likelihood on \mathbf{b} and $\boldsymbol{\theta}_k^{(z \rightarrow y)}$ are very similar to the gradients in logistic regression,

$$\nabla_{\boldsymbol{\theta}_k^{(z \rightarrow y)}} \ell^{(i)} = \left[\frac{\partial \ell^{(i)}}{\partial \theta_{k,1}^{(z \rightarrow y)}}, \frac{\partial \ell^{(i)}}{\partial \theta_{k,2}^{(z \rightarrow y)}}, \dots, \frac{\partial \ell^{(i)}}{\partial \theta_{k,K_y}^{(z \rightarrow y)}} \right]^\top \quad [3.27]$$

$$\frac{\partial \ell^{(i)}}{\partial \theta_{k,j}^{(z \rightarrow y)}} = - \frac{\partial}{\partial \theta_{k,j}^{(z \rightarrow y)}} \left(\boldsymbol{\theta}_{y^{(i)}}^{(z \rightarrow y)} \cdot \mathbf{z} - \log \sum_{y \in \mathcal{Y}} \exp \boldsymbol{\theta}_y^{(z \rightarrow y)} \cdot \mathbf{z} \right) \quad [3.28]$$

$$= \left(\Pr(y = j \mid \mathbf{z}; \Theta^{(z \rightarrow y)}, \mathbf{b}) - \delta(j = y^{(i)}) \right) z_k, \quad [3.29]$$

where $\delta(j = y^{(i)})$ is a function that returns one when $j = y^{(i)}$, and zero otherwise. The gradient $\nabla_{\mathbf{b}} \ell^{(i)}$ is similar to Equation 3.29.

The gradients on the input layer weights $\Theta^{(x \rightarrow z)}$ can be obtained by applying the chain rule of differentiation:

$$\frac{\partial \ell^{(i)}}{\partial \theta_{n,k}^{(x \rightarrow z)}} = \frac{\partial \ell^{(i)}}{\partial z_k} \frac{\partial z_k}{\partial \theta_{n,k}^{(x \rightarrow z)}} \quad [3.30]$$

$$= \frac{\partial \ell^{(i)}}{\partial z_k} \frac{\partial f(\boldsymbol{\theta}_k^{(x \rightarrow z)} \cdot \mathbf{x})}{\partial \theta_{n,k}^{(x \rightarrow z)}} \quad [3.31]$$

$$= \frac{\partial \ell^{(i)}}{\partial z_k} \times f'(\boldsymbol{\theta}_k^{(x \rightarrow z)} \cdot \mathbf{x}) \times x_n, \quad [3.32]$$

where $f'(\boldsymbol{\theta}_k^{(x \rightarrow z)} \cdot \mathbf{x})$ is the derivative of the activation function f , applied at the input

$\theta_k^{(x \rightarrow z)} \cdot \mathbf{x}$. For example, if f is the sigmoid function, then the derivative is,

$$\frac{\partial \ell^{(i)}}{\partial \theta_{n,k}^{(x \rightarrow z)}} = \frac{\partial \ell^{(i)}}{\partial z_k} \times \sigma(\theta_k^{(x \rightarrow z)} \cdot \mathbf{x}) \times (1 - \sigma(\theta_k^{(x \rightarrow z)} \cdot \mathbf{x})) \times x_n \quad [3.33]$$

$$= \frac{\partial \ell^{(i)}}{\partial z_k} \times z_k \times (1 - z_k) \times x_n. \quad [3.34]$$

1492 For intuition, consider each of the terms in the product.

- 1493 • If the negative log-likelihood $\ell^{(i)}$ does not depend much on z_k , $\frac{\partial \ell^{(i)}}{\partial z_k} \rightarrow 0$, then it
1494 doesn't matter how z_k is computed, and so $\frac{\partial \ell^{(i)}}{\partial \theta_{n,k}^{(x \rightarrow z)}} \rightarrow 0$.
- 1495 • If z_k is near 1 or 0, then the curve of the sigmoid function (Figure 3.2) is nearly flat,
1496 and changing the inputs will make little local difference. The term $z_k \times (1 - z_k)$ is
1497 maximized at $z_k = \frac{1}{2}$, where the slope of the sigmoid function is steepest.
- 1498 • If $x_n = 0$, then it does not matter how we set the weights $\theta_{n,k}^{(x \rightarrow z)}$, so $\frac{\partial \ell^{(i)}}{\partial \theta_{n,k}^{(x \rightarrow z)}} = 0$.

1499 3.3.1 Backpropagation

1500 In the equations above, the value $\frac{\partial \ell^{(i)}}{\partial z_k}$ is reused in the derivatives with respect to each
1501 $\theta_{n,k}^{(x \rightarrow z)}$. It should therefore be computed once, and then cached. Furthermore, we should
1502 only compute any derivative once we have already computed all of the necessary "inputs"
1503 demanded by the chain rule of differentiation. This combination of sequencing, caching,
1504 and differentiation is known as **backpropagation**. It can be generalized to any directed
1505 acyclic **computation graph**.

1506 A computation graph is a declarative representation of a computational process. At
1507 each node t , compute a value v_t by applying a function f_t to a (possibly empty) list of
1508 parent nodes, π_t . For example, in a feedforward network with one hidden layer, there are
1509 nodes for the input $\mathbf{x}^{(i)}$, the hidden layer \mathbf{z} , the predicted output $\tilde{\mathbf{y}}$, and the parameters
1510 $\{\Theta^{(x \rightarrow z)}, \Theta^{(z \rightarrow y)}, \mathbf{b}\}$. During training, there is also a node for the observed label $y^{(i)}$ and
1511 the loss $\ell^{(i)}$. Computation graphs have three main types of nodes:

1512 **Variables.** The variables include the *inputs* \mathbf{x} , the *hidden nodes* \mathbf{z} , the outputs \mathbf{y} , and the
1513 loss function. Inputs are variables that do not have parents. Backpropagation com-
1514putes the gradients with respect to all variables except the inputs, but does not up-
1515 date the variables during learning.

1516 **Parameters.** In a feedforward network, the parameters include the weights and offsets.
1517 Parameter nodes do not have parents, and they are updated during learning.

Algorithm 6 General backpropagation algorithm. In the computation graph G , every node contains a function f_t and a set of parent nodes π_t ; the inputs to the graph are $x^{(i)}$.

```

1: procedure BACKPROP( $G = \{f_t, \pi_t\}_{t=1}^T, x^{(i)}$ )
2:    $v_{t(n)} \leftarrow x_n^{(i)}$  for all  $n$  and associated computation nodes  $t(n)$ .
3:   for  $t \in \text{TOPLOGICALSORT}(G)$  do  $\triangleright$  Forward pass: compute value at each node
4:     if  $|\pi_t| > 0$  then
5:        $v_t \leftarrow f_t(v_{\pi_{t,1}}, v_{\pi_{t,2}}, \dots, v_{\pi_{t,N_t}})$ 
6:      $g_{\text{objective}} = 1$   $\triangleright$  Backward pass: compute gradients at each node
7:     for  $t \in \text{REVERSE}(\text{TOPLOGICALSORT}(G))$  do
8:        $g_t \leftarrow \sum_{t': t \in \pi_{t'}} g_{t'} \times \nabla_{v_t} v_{t'}$   $\triangleright$  Sum over all  $t'$  that are children of  $t$ , propagating
        the gradient  $g_{t'}$ , scaled by the local gradient  $\nabla_{v_t} v_{t'}$ 
9:   return  $\{g_1, g_2, \dots, g_T\}$ 

```

1518 **Objective.** The *objective* node is not the parent of any other node. Backpropagation begins
 1519 by computing the gradient with respect to this node.

1520 If the computation graph is a directed acyclic graph, then it is possible to order the
 1521 nodes with a topological sort, so that if node t is a parent of node t' , then $t < t'$. This
 1522 means that the values $\{v_t\}_{t=1}^T$ can be computed in a single forward pass. The topolog-
 1523 ical sort is reversed when computing gradients: each gradient g_t is computed from the
 1524 gradients of the children of t , implementing the chain rule of differentiation. The general
 1525 backpropagation algorithm for computation graphs is shown in Algorithm 6, and illus-
 1526 trated in Figure 3.3.

1527 While the gradients with respect to each parameter may be complex, they are com-
 1528 posed of products of simple parts. For many networks, all gradients can be computed
 1529 through **automatic differentiation**. This means that end users need only specify the feed-
 1530 forward computation, and the gradients necessary for learning can be obtained automati-
 1531 cally. There are many software libraries that perform automatic differentiation on compu-
 1532 tation graphs, such as Torch (Collobert et al., 2011), TensorFlow (Abadi et al., 2016), and
 1533 DyNet (Neubig et al., 2017). One important distinction between these libraries is whether
 1534 they support **dynamic computation graphs**, in which the structure of the computation
 1535 graph varies across instances. Static computation graphs are compiled in advance, and
 1536 can be applied to fixed-dimensional data, such as bag-of-words vectors. In many natu-
 1537 ral language processing problems, each input has a distinct structure, requiring a unique
 1538 computation graph.

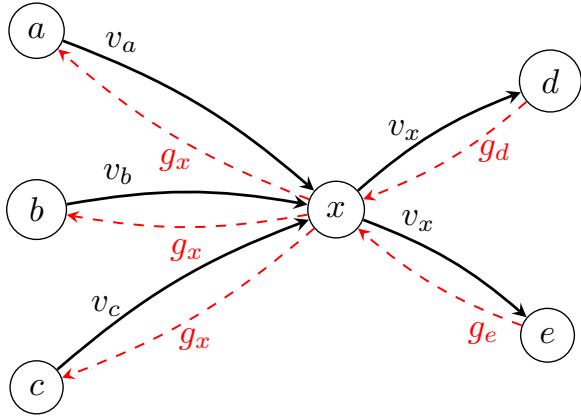


Figure 3.3: Backpropagation at a single node x in the computation graph. The values of the predecessors v_a, v_b, v_c are the inputs to x , which computes v_x , and passes it on to the successors d and e . The gradients at the successors g_d and g_e are passed back to x , where they are incorporated into the gradient g_x , which is then passed back to the predecessors a, b , and c .

1539 3.3.2 Regularization and dropout

1540 In linear classification, overfitting was addressed by augmenting the objective with a reg-
 1541 ularization term, $\lambda \|\theta\|_2^2$. This same approach can be applied to feedforward neural net-
 1542 works, penalizing each matrix of weights:

$$L = \sum_{i=1}^N \ell^{(i)} + \lambda_{z \rightarrow y} \|\Theta^{(z \rightarrow y)}\|_F^2 + \lambda_{x \rightarrow z} \|\Theta^{(x \rightarrow z)}\|_F^2, \quad [3.35]$$

1543 where $\|\Theta\|_F^2 = \sum_{i,j} \theta_{i,j}^2$ is the squared **Frobenius norm**, which generalizes the L_2 norm
 1544 to matrices. The bias parameters b are not regularized, as they do not contribute to the
 1545 sensitivity of the classifier to the inputs. In gradient-based optimization, the practical
 1546 effect of Frobenius norm regularization is that the weights “decay” towards zero at each
 1547 update, motivating the alternative name **weight decay**.

1548 Another approach to controlling model complexity is **dropout**, which involves ran-
 1549 domly setting some computation nodes to zero during training (Srivastava et al., 2014).
 1550 For example, in the feedforward network, on each training instance, with probability ρ we
 1551 set each input x_n and each hidden layer node z_k to zero. Srivastava et al. (2014) recom-
 1552 mend $\rho = 0.5$ for hidden units, and $\rho = 0.2$ for input units. Dropout is also incorpo-
 1553 rated in the gradient computation, so if node z_k is dropped, then none of the weights $\theta_k^{(x \rightarrow z)}$ will
 1554 be updated for this instance. Dropout prevents the network from learning to depend too
 1555 much on any one feature or hidden node, and prevents **feature co-adaptation**, in which a

hidden unit is only useful in combination with one or more other hidden units. Dropout is a special case of **feature noising**, which can also involve adding Gaussian noise to inputs or hidden units (Holmstrom and Koistinen, 1992). Wager et al. (2013) show that dropout is approximately equivalent to “adaptive” L_2 regularization, with a separate regularization penalty for each feature.

3.3.3 *Learning theory

Chapter 2 emphasized the importance of **convexity** for learning: for convex objectives, the global optimum can be found efficiently. The negative log-likelihood and hinge loss are convex functions of the parameters of the output layer. However, the output of a feed-forward network is generally not a convex function of the parameters of the input layer, $\Theta^{(x \rightarrow z)}$. Feedforward networks can be viewed as function composition, where each layer is a function that is applied to the output of the previous layer. Convexity is generally not preserved in the composition of two convex functions — and furthermore, “squashing” activation functions like tanh and sigmoid are not convex.

The non-convexity of hidden layer neural networks can also be seen by permuting the elements of the hidden layer, from $z = [z_1, z_2, \dots, z_{K_z}]$ to $\tilde{z} = [z_{\pi(1)}, z_{\pi(2)}, \dots, z_{\pi(K_z)}]$. This corresponds to applying π to the rows of $\Theta^{(x \rightarrow z)}$ and the columns of $\Theta^{(z \rightarrow y)}$, resulting in permuted parameter matrices $\Theta_\pi^{(x \rightarrow z)}$ and $\Theta_\pi^{(z \rightarrow y)}$. As long as this permutation is applied consistently, the loss will be identical, $L(\Theta) = L(\Theta_\pi)$: it is *invariant* to this permutation. However, the loss of the linear combination $L(\alpha\Theta + (1 - \alpha)\Theta_\pi)$ will generally not be identical to the loss under Θ or its permutations. If $L(\Theta)$ is better than the loss at any points in the immediate vicinity, and if $L(\Theta) = L(\Theta_\pi)$, then the loss function does not satisfy the definition of convexity (see § 2.3). One of the exercises asks you to prove this more rigorously.

In practice, the existence of multiple optima is not necessarily problematic, if all such optima are permutations of the sort described in the previous paragraph. In contrast, “bad” local optima are better than their neighbors, but much worse than the global optimum. Fortunately, in large feedforward neural networks, most local optima are nearly as good as the global optimum (Choromanska et al., 2015), which helps to explain why back-propagation works in practice. More generally, a **critical point** is one at which the gradient is zero. Critical points may be local optima, but they may also be **saddle points**, which are local minima in some directions, but local *maxima* in other directions. For example, the equation $x_1^2 - x_2^2$ has a saddle point at $x = (0, 0)$.⁴ In large networks, the overwhelming majority of critical points are saddle points, rather than local minima or maxima (Dauphin et al., 2014). Saddle points can pose problems for gradient-based optimization, since learning will slow to a crawl as the gradient goes to zero. However, the noise introduced by

⁴Thanks to Rong Ge’s blogpost for this example, <http://www.offconvex.org/2016/03/22/saddlepoints/>

1592 stochastic gradient descent, and by feature noising techniques such as dropout, can help
 1593 online optimization to escape saddle points and find high-quality optima (Ge et al., 2015).
 1594 Other techniques address saddle points directly, using local reconstructions of the Hessian
 1595 matrix (Dauphin et al., 2014) or higher-order derivatives (Anandkumar and Ge, 2016).

1596 **3.3.4 Tricks**

1597 Getting neural networks to work effectively sometimes requires heuristic “tricks” (Bottou,
 1598 2012; Goodfellow et al., 2016; Goldberg, 2017b). This section presents some tricks that are
 1599 especially important.

Initialization Initialization is not especially important for linear classifiers, since convexity ensures that the global optimum can usually be found quickly. But for multilayer neural networks, it is helpful to have a good starting point. One reason is that if the magnitude of the initial weights is too large, a sigmoid or tanh nonlinearity will be saturated, leading to a small gradient, and slow learning. Large gradients are also problematic. Initialization can help avoid these problems, by ensuring that the variance over the initial gradients is constant and bounded throughout the network. For networks with tanh activation functions, this can be achieved by sampling the initial weights from the following uniform distribution (Glorot and Bengio, 2010),

$$\theta_{i,j} \sim U \left[-\frac{\sqrt{6}}{\sqrt{d_{\text{in}}(n) + d_{\text{out}}(n)}}, \frac{\sqrt{6}}{\sqrt{d_{\text{in}}(n) + d_{\text{out}}(n)}} \right], \quad [3.36]$$

[3.37]

1600 For the weights leading to a ReLU activation function, He et al. (2015) use similar argu-
 1601 mentation to justify sampling from a zero-mean Gaussian distribution,

$$\theta_{i,j} \sim N(0, \sqrt{2/d_{\text{in}}(n)}) \quad [3.38]$$

Rather than initializing the weights independently, it can be beneficial to initialize each layer jointly as an **orthonormal matrix**, ensuring that $\Theta^\top \Theta = \mathbb{I}$ (Saxe et al., 2014). Orthonormal matrices preserve the norm of the input, so that $\|\Theta x\| = \|x\|$, which prevents the gradients from exploding or vanishing. Orthogonality ensures that the hidden units are uncorrelated, so that they correspond to different features of the input. Orthonormal initialization can be performed by applying **singular value decomposition** to a matrix of

values sampled from a standard normal distribution:

$$a_{i,j} \sim N(0, 1) \quad [3.39]$$

$$\mathbf{A} = \{a_{i,j}\}_{i=1,j=1}^{d_{\text{in}}(j), d_{\text{out}}(j)} \quad [3.40]$$

$$\mathbf{U}, \mathbf{S}, \mathbf{V}^\top = \text{SVD}(\mathbf{A}) \quad [3.41]$$

$$\Theta^{(j)} \leftarrow \mathbf{U}. \quad [3.42]$$

1602 The matrix \mathbf{U} contains the **singular vectors** of \mathbf{A} , and is guaranteed to be orthonormal.
 1603 For more on singular value decomposition, see chapter 14.

1604 Even with careful initialization, there can still be significant variance in the final re-
 1605 sults. It can be useful to make multiple training runs, and select the one with the best
 1606 performance on a heldout development set.

1607 **Clipping and normalizing the gradients** As already discussed, the magnitude of the
 1608 gradient can pose problems for learning: too large, and learning can diverge, with suc-
 1609 cessive updates thrashing between increasingly extreme values; too small, and learning can
 1610 grind to a halt. Several heuristics have been proposed to address this issue.

1611 • In **gradient clipping** (Pascanu et al., 2013), an upper limit is placed on the norm of
 1612 the gradient, and the gradient is rescaled when this limit is exceeded,

$$\text{CLIP}(\hat{\mathbf{g}}) = \begin{cases} \mathbf{g} & \|\hat{\mathbf{g}}\| < \tau \\ \frac{\tau}{\|\mathbf{g}\|} \mathbf{g} & \text{otherwise.} \end{cases} \quad [3.43]$$

1613 • In **batch normalization** (Ioffe and Szegedy, 2015), the inputs to each computation
 1614 node are recentered by their mean and variance across all of the instances in the
 minibatch \mathcal{B} (see § 2.5.2). For example, in a feedforward network with one hidden
 layer, batch normalization would transform the inputs to the hidden layer as follows:

$$\boldsymbol{\mu}^{(\mathcal{B})} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathbf{x}^{(i)} \quad [3.44]$$

$$\mathbf{s}^{(\mathcal{B})} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} (\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(\mathcal{B})})^2 \quad [3.45]$$

$$\bar{\mathbf{x}}^{(i)} = (\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(\mathcal{B})}) / \sqrt{\mathbf{s}^{(\mathcal{B})}}. \quad [3.46]$$

1613 Empirically, this speeds convergence of deep architectures. One explanation is that
 1614 it helps to correct for changes in the distribution of activations during training.

- In **layer normalization** (Ba et al., 2016), the inputs to each nonlinear activation function are recentered across the layer:

$$\mathbf{a} = \Theta^{(x \rightarrow z)} \mathbf{x} \quad [3.47]$$

$$\mu = \frac{1}{K_z} \sum_{k=1}^{K_z} a_k \quad [3.48]$$

$$s = \frac{1}{K_z} \sum_{k=1}^{K_z} (a_k - \mu)^2 \quad [3.49]$$

$$\mathbf{z} = (\mathbf{a} - \mu) / \sqrt{s}. \quad [3.50]$$

1615 Layer normalization has similar motivations to batch normalization, but it can be
 1616 applied across a wider range of architectures and training conditions.

Online optimization The trend towards deep learning has spawned a cottage industry of online optimization algorithms, which attempt to improve on stochastic gradient descent. **AdaGrad** was reviewed in § 2.5.2; its main innovation is to set adaptive learning rates for each parameter by storing the sum of squared gradients. Rather than using the sum over the entire training history, we can keep a running estimate,

$$v_j^{(t)} = \beta v_j^{(t-1)} + (1 - \beta) g_{t,j}^2, \quad [3.51]$$

1617 where $g_{t,j}$ is the gradient with respect to parameter j at time t , and $\beta \in [0, 1]$. This term
 1618 places more emphasis on recent gradients, and is employed in the **AdaDelta** (Zeiler, 2012)
 1619 and **Adam** (Kingma and Ba, 2014) optimizers. Online optimization and its theoretical
 1620 background are reviewed by Bottou et al. (2016). **Early stopping**, mentioned in § 2.2.2,
 1621 can help to avoid overfitting, by terminating training after reaching a plateau in the per-
 1622 formance on a heldout validation set.

1623 3.4 Convolutional neural networks

1624 A basic weakness of the bag-of-words model is its inability to account for the ways in
 1625 which words combine to create meaning, including even simple reversals such as *not*
 1626 *pleasant, hardly a generous offer*, and *I wouldn't mind missing the flight*. Similarly, computer
 1627 vision faces the challenge of identifying the semantics of images from pixel features that
 1628 are uninformative in isolation. An earlier generation of computer vision research fo-
 1629 cused on designing *filters* to aggregate local pixel-level features into more meaningful
 1630 representations, such as edges and corners (e.g., Canny, 1987). Similarly, earlier NLP re-
 1631 search attempted to capture multiword linguistic phenomena by hand-designed lexical
 1632 patterns (Hobbs et al., 1997). In both cases, the output of the filters and patterns could

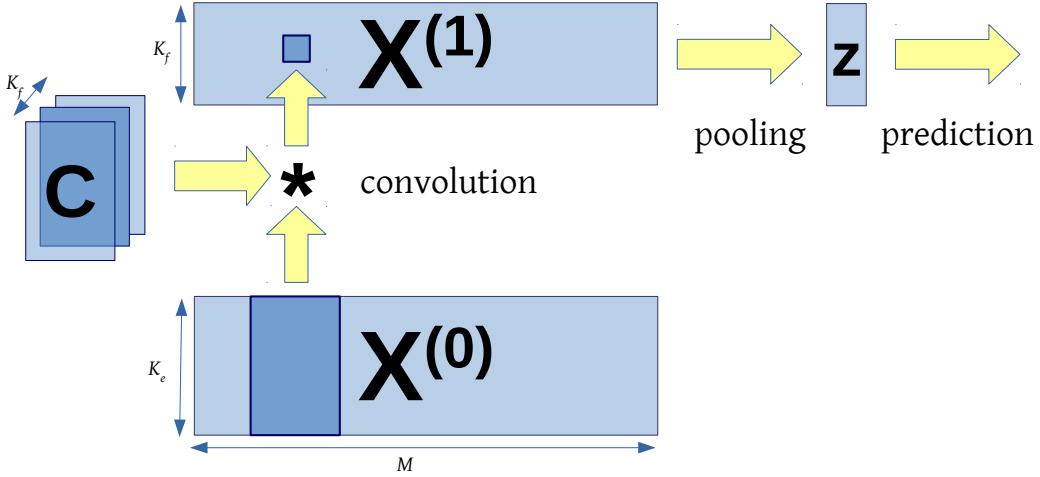


Figure 3.4: A convolutional neural network for text classification

then act as base features in a linear classifier. But rather than designing these feature extractors by hand, a better approach is to learn them, using the magic of backpropagation. This is the idea behind **convolutional neural networks**.

Following § 3.2.4, define the base layer of a neural network as,

$$\mathbf{X}^{(0)} = \Theta^{(x \rightarrow z)}[\mathbf{e}_{w_1}, \mathbf{e}_{w_2}, \dots, \mathbf{e}_{w_M}], \quad [3.52]$$

where \mathbf{e}_{w_m} is a column vector of zeros, with a 1 at position w_m . The base layer has dimension $\mathbf{X}^{(0)} \in \mathbb{R}^{K_e \times M}$, where K_e is the size of the word embeddings. To merge information across adjacent words, we *convolve* $\mathbf{X}^{(0)}$ with a set of filter matrices $\mathbf{C}^{(k)} \in \mathbb{R}^{K_e \times h}$. Convolution is indicated by the symbol $*$, and is defined,

$$\mathbf{X}^{(1)} = f(\mathbf{b} + \mathbf{C} * \mathbf{X}^{(0)}) \implies x_{k,m}^{(1)} = f \left(b_k + \sum_{k'=1}^{K_e} \sum_{n=1}^h c_{k',n}^{(k)} \times x_{k',m+n-1}^{(0)} \right), \quad [3.53]$$

where f is an activation function such as tanh or ReLU, and \mathbf{b} is a vector of offsets. The convolution operation slides the matrix $\mathbf{C}^{(k)}$ across the columns of $\mathbf{X}^{(0)}$; at each position m , compute the elementwise product $\mathbf{C}^{(k)} \odot \mathbf{X}_{m:m+h-1}^{(0)}$, and take the sum.

A simple filter might compute a weighted average over nearby words,

$$\mathbf{C}^{(k)} = \begin{bmatrix} 0.5 & 1 & 0.5 \\ 0.5 & 1 & 0.5 \\ \dots & \dots & \dots \\ 0.5 & 1 & 0.5 \end{bmatrix}, \quad [3.54]$$

1641 thereby representing trigram units like *not so unpleasant*. In **one-dimensional convolution**,
 1642 each filter matrix $\mathbf{C}^{(k)}$ is constrained to have non-zero values only at row k (Kalchbrenner et al., 2014).

1644 To deal with the beginning and end of the input, the base matrix $\mathbf{X}^{(0)}$ may be padded
 1645 with h column vectors of zeros at the beginning and end; this is known as **wide convolution**. If padding is not applied, then the output from each layer will be $h - 1$ units smaller
 1646 than the input; this is known as **narrow convolution**. The filter matrices need not have
 1647 identical filter widths, so more generally we could write h_k to indicate width of filter
 1648 $\mathbf{C}^{(k)}$. As suggested by the notation $\mathbf{X}^{(0)}$, multiple layers of convolution may be applied,
 1649 so that $\mathbf{X}^{(d)}$ is the input to $\mathbf{X}^{(d+1)}$.

After D convolutional layers, we obtain a matrix representation of the document $\mathbf{X}^{(D)} \in \mathbb{R}^{K_z \times M}$. If the instances have variable lengths, it is necessary to aggregate over all M word positions to obtain a fixed-length representation. This can be done by a **pooling** operation, such as max-pooling (Collobert et al., 2011) or average-pooling,

$$z = \text{MaxPool}(\mathbf{X}^{(D)}) \implies z_k = \max(x_{k,1}^{(D)}, x_{k,2}^{(D)}, \dots, x_{k,M}^{(D)}) \quad [3.55]$$

$$z = \text{AvgPool}(\mathbf{X}^{(D)}) \implies z_k = \frac{1}{M} \sum_{m=1}^M x_{k,m}^{(D)}. \quad [3.56]$$

1651 The vector z can now act as a layer in a feedforward network, culminating in a prediction
 1652 \hat{y} and a loss $\ell^{(i)}$. The setup is shown in Figure 3.4.

Just as in feedforward networks, the parameters $(\mathbf{C}^{(k)}, \mathbf{b}, \Theta)$ can be learned by backpropagating from the classification loss. This requires backpropagating through the max-pooling operation, which is a discontinuous function of the input. But because we need only a local gradient, backpropagation flows only through the argmax m :

$$\frac{\partial z_k}{\partial x_{k,m}^{(D)}} = \begin{cases} 1, & x_{k,m}^{(D)} = \max(x_{k,1}^{(D)}, x_{k,2}^{(D)}, \dots, x_{k,M}^{(D)}) \\ 0, & \text{otherwise.} \end{cases} \quad [3.57]$$

1653 The computer vision literature has produced a huge variety of convolutional architectures,
 1654 and many of these bells and whistles can be applied to text data. One avenue for
 1655 improvement is more complex pooling operations, such as k -max pooling (Kalchbrenner
 1656 et al., 2014), which returns a matrix of the k largest values for each filter. Another innovation
 1657 is the use of **dilated convolution** to build multiscale representations (Yu and Koltun,
 1658 2016). At each layer, the convolutional operator applied in *strides*, skipping ahead by s
 1659 steps after each feature. As we move up the hierarchy, each layer is s times smaller than
 1660 the layer below it, effectively summarizing the input. This idea is shown in Figure 3.5.
 1661 Multi-layer convolutional networks can also be augmented with “shortcut” connections,
 1662 as in the ResNet model from § 3.2.2 (Johnson and Zhang, 2017).

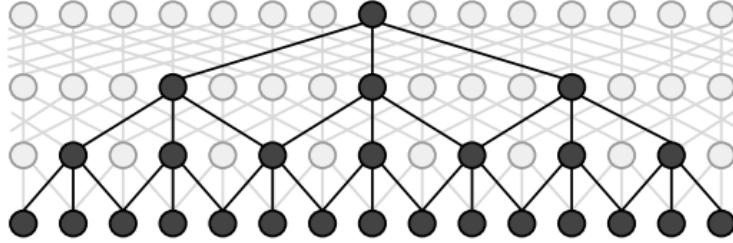


Figure 3.5: A dilated convolutional neural network captures progressively larger context through recursive application of the convolutional operator (Strubell et al., 2017) [todo: permission]

1663 Additional resources

1664 The deep learning textbook by Goodfellow et al. (2016) covers many of the topics in this
 1665 chapter in more detail. For a comprehensive review of neural networks in natural lan-
 1666 guage processing, see (Goldberg, 2017b). A seminal work on deep learning in natural
 1667 language processing is the aggressively titled “Natural Language Processing (Almost)
 1668 from Scratch”, which uses convolutional neural networks to perform a range of language
 1669 processing tasks (Collobert et al., 2011). This chapter focuses on feedforward and con-
 1670 volutional neural networks, but recurrent neural networks are one of the most important
 1671 deep learning architectures for natural language processing. They are covered extensively
 1672 in chapters 6 and 7.

1673 The role of deep learning in natural language processing research has caused angst
 1674 in some parts of the natural language processing research community (e.g., Goldberg,
 1675 2017a), especially as some of the more zealous deep learning advocates have argued that
 1676 end-to-end learning from “raw” text can eliminate the need for linguistic constructs such
 1677 as sentences, phrases, and even words (Zhang et al., 2015, originally titled *Text understand-
 1678 ing from scratch*). These developments were surveyed by Manning (2016).

1679 Exercises

- 1680 1. Prove that the softmax and sigmoid functions are equivalent when the number of
 1681 possible labels is two. Specifically, for any $\Theta^{(z \rightarrow y)}$ (omitting the offset b for sim-
 1682 plicity), show how to construct a vector of weights θ such that,

$$\text{SoftMax}(\Theta^{(z \rightarrow y)} z)[0] = \sigma(\theta \cdot z). \quad [3.58]$$

1683 2. Design a feedforward network to compute the XOR function:

$$f(x_1, x_2) = \begin{cases} -1, & x_1 = 1, x_2 = 1 \\ 1, & x_1 = 1, x_2 = 0 \\ 1, & x_1 = 0, x_2 = 1 \\ -1, & x_1 = 0, x_2 = 0 \end{cases}. \quad [3.59]$$

1684 Your network should have a single output node which uses the Sign activation function.
 1685 Use a single hidden layer, with ReLU activation functions. Describe all weights
 1686 and offsets.

- 1687 3. Consider the same network as above (with ReLU activations for the hidden layer),
 1688 with an arbitrary differentiable loss function $\ell(y^{(i)}, \tilde{y})$, where \tilde{y} is the activation of
 1689 the output node. Suppose all weights and offsets are initialized to zero. Prove that
 1690 gradient-based optimization cannot learn the desired function from this initializa-
 1691 tion.
- 1692 4. The simplest solution to the previous problem relies on the use of the ReLU activa-
 1693 tion function at the hidden layer. Now consider a network with arbitrary activations
 1694 on the hidden layer. Show that if the initial weights are any uniform constant, then
 1695 it is not possible to learn the desired function.

- 1696 5. Consider a network in which: the base features are all binary, $\mathbf{x} \in \{0, 1\}^M$; the
 1697 hidden layer activation function is sigmoid, $z_k = \sigma(\theta_k \cdot \mathbf{x})$; and the initial weights
 1698 are sampled independently from a standard normal distribution, $\theta_{j,k} \sim N(0, 1)$.

- 1699 • Show how the probability of a small initial gradient on any weight, $\frac{\partial z_k}{\partial \theta_{j,k}} < \alpha$,
 1700 depends on the size of the input M . **Hint:** use the lower bound,

$$\Pr(\sigma(\theta_k \cdot \mathbf{x}) \times (1 - \sigma(\theta_k \cdot \mathbf{x})) < \alpha) \geq 2 \Pr(\sigma(\theta_k \cdot \mathbf{x}) < \alpha), \quad [3.60]$$

1701 and relate this probability to the variance $V[\theta_k \cdot \mathbf{x}]$.

- 1702 • Design an alternative initialization that removes this dependence.

- 1703 6. Suppose that the parameters $\Theta = \{\Theta^{(x \rightarrow z)}, \Theta(z \rightarrow y), \mathbf{b}\}$ are a local optimum of a
 1704 feedforward network in the following sense: there exists some $\epsilon > 0$ such that,

$$\begin{aligned} & \left(\|\tilde{\Theta}^{(x \rightarrow z)} - \Theta^{(x \rightarrow z)}\|_F^2 + \|\tilde{\Theta}^{(z \rightarrow y)} - \Theta^{(z \rightarrow y)}\|_F^2 + \|\tilde{\mathbf{b}} - \mathbf{b}\|_2^2 < \epsilon \right) \\ & \Rightarrow \left(L(\tilde{\Theta}) > L(\Theta) \right) \end{aligned} \quad [3.61]$$

1705 Define the function π as a permutation on the hidden units, as described in § 3.3.3,
 1706 so that for any Θ , $L(\Theta) = L(\Theta_\pi)$. Prove that if a feedforward network has a local
 optimum in the sense of Equation 3.61, then its loss is not a convex function of the
 parameters Θ , using the definition of convexity from § 2.3

1707 **Chapter 4**

1708 **Linguistic applications of
1709 classification**

1710 Having learned some techniques for classification, this chapter shifts the focus from math-
1711 ematics to linguistic applications. Later in the chapter, we will consider the design deci-
1712 sions involved in text classification, as well as evaluation practices.

1713 **4.1 Sentiment and opinion analysis**

1714 A popular application of text classification is to automatically determine the **sentiment**
1715 or **opinion polarity** of documents such as product reviews and social media posts. For
1716 example, marketers are interested to know how people respond to advertisements, ser-
1717 vices, and products (Hu and Liu, 2004); social scientists are interested in how emotions
1718 are affected by phenomena such as the weather (Hannak et al., 2012), and how both opin-
1719 ions and emotions spread over social networks (Coviello et al., 2014; Miller et al., 2011).
1720 In the field of **digital humanities**, literary scholars track plot structures through the flow
1721 of sentiment across a novel (Jockers, 2015).¹

1722 Sentiment analysis can be framed as a direct application of document classification,
1723 assuming reliable labels can be obtained. In the simplest case, sentiment analysis is a
1724 two or three-class problem, with sentiments of POSITIVE, NEGATIVE, and possibly NEU-
1725 TRAL. Such annotations could be annotated by hand, or obtained automatically through
1726 a variety of means:

- 1727 • Tweets containing happy emoticons can be marked as positive, sad emoticons as
1728 negative (Read, 2005; Pak and Paroubek, 2010).

¹Comprehensive surveys on sentiment analysis and related problems are offered by Pang and Lee (2008) and Liu (2015).

- 1729 • Reviews with four or more stars can be marked as positive, two or fewer stars as
 1730 negative (Pang et al., 2002).
- 1731 • Statements from politicians who are voting for a given bill are marked as positive
 1732 (towards that bill); statements from politicians voting against the bill are marked as
 1733 negative (Thomas et al., 2006).

1734 The bag-of-words model is a good fit for sentiment analysis at the document level: if
 1735 the document is long enough, we would expect the words associated with its true senti-
 1736 ment to overwhelm the others. Indeed, **lexicon-based sentiment analysis** avoids machine
 1737 learning altogether, and classifies documents by counting words against positive and neg-
 1738 ative sentiment word lists (Taboada et al., 2011).

1739 Lexicon-based classification is less effective for short documents, such as single-sentence
 1740 reviews or social media posts. In these documents, linguistic issues like **negation** and **ir-**
 1741 **realis** (Polanyi and Zaenen, 2006) — events that are hypothetical or otherwise non-factual
 1742 — can make bag-of-words classification ineffective. Consider the following examples:

- 1743 (4.1) That's not bad for the first day.
- 1744 (4.2) This is not the worst thing that can happen.
- 1745 (4.3) It would be nice if you acted like you understood.
- 1746 (4.4) There is no reason at all to believe that the polluters are suddenly going to be-
 1747 come reasonable. (Wilson et al., 2005)
- 1748 (4.5) This film should be brilliant. The actors are first grade. Stallone plays a happy,
 1749 wonderful man. His sweet wife is beautiful and adores him. He has a fascinat-
 1750 ing gift for living life fully. It sounds like a great plot, **however**, the film is a
 1751 failure. (Pang et al., 2002)

1752 A minimal solution is to move from a bag-of-words model to a bag-of-**bigrams** model,
 1753 where each base feature is a pair of adjacent words, e.g.,

$$(that's, not), (not, bad), (bad, for), \dots \quad [4.1]$$

1754 Bigrams can handle relatively straightforward cases, such as when an adjective is immedi-
 1755 ately negated; trigrams would be required to extend to larger contexts (e.g., *not the worst*).
 1756 But this approach will not scale to more complex examples like (4.4) and (4.5). More
 1757 sophisticated solutions try to account for the syntactic structure of the sentence (Wilson
 1758 et al., 2005; Socher et al., 2013), or apply more complex classifiers such as **convolutional**
 1759 **neural networks** (Kim, 2014), which are described in chapter 3.

1760 **4.1.1 Related problems**

1761 **Subjectivity** Closely related to sentiment analysis is **subjectivity detection**, which re-
1762 quires identifying the parts of a text that express subjective opinions, as well as other non-
1763 factual content such as speculation and hypotheticals (Riloff and Wiebe, 2003). This can be
1764 done by treating each sentence as a separate document, and then applying a bag-of-words
1765 classifier: indeed, Pang and Lee (2004) do exactly this, using a training set consisting of
1766 (mostly) subjective sentences gathered from movie reviews, and (mostly) objective sen-
1767 tences gathered from plot descriptions. They augment this bag-of-words model with a
1768 graph-based algorithm that encourages nearby sentences to have the same subjectivity
1769 label.

1770 **Stance classification** In debates, each participant takes a side: for example, advocating
1771 for or against proposals like adopting a vegetarian lifestyle or mandating free college ed-
1772 ucation. The problem of stance classification is to identify the author’s position from the
1773 text of the argument. In some cases, there is training data available for each position,
1774 so that standard document classification techniques can be employed. In other cases, it
1775 suffices to classify each document as whether it is in support or opposition of the argu-
1776 ment advanced by a previous document (Anand et al., 2011). In the most challenging
1777 case, there is no labeled data for any of the stances, so the only possibility is group docu-
1778 ments that advocate the same position (Somasundaran and Wiebe, 2009). This is a form
1779 of **unsupervised learning**, discussed in chapter 5.

1780 **Targeted sentiment analysis** The expression of sentiment is often more nuanced than a
1781 simple binary label. Consider the following examples:

1782 (4.6) The vodka was good, but the meat was rotten.

1783 (4.7) Go to Heaven for the climate, Hell for the company. —Mark Twain

1784 These statements display a mixed overall sentiment: positive towards some entities (e.g.,
1785 *the vodka*), negative towards others (e.g., *the meat*). **Targeted sentiment analysis** seeks to
1786 identify the writer’s sentiment towards specific entities (Jiang et al., 2011). This requires
1787 identifying the entities in the text and linking them to specific sentiment words — much
1788 more than we can do with the classification-based approaches discussed thus far. For
1789 example, Kim and Hovy (2006) analyze sentence-internal structure to determine the topic
1790 of each sentiment expression.

1791 **Aspect-based opinion mining** seeks to identify the sentiment of the author of a review
1792 towards predefined aspects such as PRICE and SERVICE, or, in the case of (4.7), CLIMATE
1793 and COMPANY (Hu and Liu, 2004). If the aspects are not defined in advance, it may again
1794 be necessary to employ **unsupervised learning** methods to identify them (e.g., Branavan
1795 et al., 2009).

1796 **Emotion classification** While sentiment analysis is framed in terms of positive and neg-
 1797 ative categories, psychologists generally regard **emotion** as more multifaceted. For ex-
 1798 ample, Ekman (1992) argues that there are six basic emotions — happiness, surprise, fear,
 1799 sadness, anger, and contempt — and that they are universal across human cultures. Alm
 1800 et al. (2005) build a linear classifier for recognizing the emotions expressed in children’s
 1801 stories. The ultimate goal of this work was to improve text-to-speech synthesis, so that
 1802 stories could be read with intonation that reflected the emotional content. They used bag-
 1803 of-words features, as well as features capturing the story type (e.g., jokes, folktales), and
 1804 structural features that reflect the position of each sentence in the story. The task is diffi-
 1805 cult: even human annotators frequently disagreed with each other, and the best classifiers
 1806 achieved accuracy between 60-70%.

1807 4.1.2 Alternative approaches to sentiment analysis

1808 **Regression** A more challenging version of sentiment analysis is to determine not just
 1809 the class of a document, but its rating on a numerical scale (Pang and Lee, 2005). If the
 1810 scale is continuous, it is most natural to apply **regression**, identifying a set of weights θ
 1811 that minimize the squared error of a predictor $\hat{y} = \theta \cdot x + b$, where b is an offset. This
 1812 approach is called **linear regression**, and sometimes **least squares**, because the regression
 1813 coefficients θ are determined by minimizing the squared error, $(y - \hat{y})^2$. If the weights are
 1814 regularized using a penalty $\lambda \|\theta\|_2^2$, then it is **ridge regression**. Unlike logistic regression,
 1815 both linear regression and ridge regression can be solved in closed form as a system of
 1816 linear equations.

1817 **Ordinal ranking** In many problems, the labels are ordered but discrete: for example,
 1818 product reviews are often integers on a scale of 1 – 5, and grades are on a scale of A – F.
 1819 Such problems can be solved by discretizing the score $\theta \cdot x$ into “ranks”,

$$\hat{y} = \underset{r: \theta \cdot x \geq b_r}{\operatorname{argmin}} r, \quad [4.2]$$

1820 where $\mathbf{b} = [b_1 = -\infty, b_2, b_3, \dots, b_K]$ is a vector of boundaries. It is possible to learn the
 1821 weights and boundaries simultaneously, using a perceptron-like algorithm (Crammer and
 1822 Singer, 2001).

1823 **Lexicon-based classification** Sentiment analysis is one of the only NLP tasks where
 1824 hand-crafted feature weights are still widely employed. In **lexicon-based classification** (Taboada
 1825 et al., 2011), the user creates a list of words for each label, and then classifies each docu-
 1826 ment based on how many of the words from each list are present. In our linear classifica-
 1827 tion framework, this is equivalent to choosing the following weights:

$$\theta_{y,j} = \begin{cases} 1, & j \in \mathcal{L}_y \\ 0, & \text{otherwise,} \end{cases} \quad [4.3]$$

1828 where \mathcal{L}_y is the lexicon for label y . Compared to the machine learning classifiers discussed
 1829 in the previous chapters, lexicon-based classification may seem primitive. However, su-
 1830 pervised machine learning relies on large annotated datasets, which are time-consuming
 1831 and expensive to produce. If the goal is to distinguish two or more categories in a new
 1832 domain, it may be simpler to start by writing down a list of words for each category.

1833 An early lexicon was the *General Inquirer* (Stone, 1966). Today, popular sentiment lex-
 1834 cons include sentiwordnet (Esuli and Sebastiani, 2006) and an evolving set of lexicons
 1835 from Liu (2015). For emotions and more fine-grained analysis, *Linguistic Inquiry and Word*
 1836 *Count* (LIWC) provides a set of lexicons (Tausczik and Pennebaker, 2010). The MPQA lex-
 1837 icon indicates the polarity (positive or negative) of 8221 terms, as well as whether they are
 1838 strongly or weakly subjective (Wiebe et al., 2005). A comprehensive comparison of senti-
 1839 ment lexicons is offered by Ribeiro et al. (2016). Given an initial **seed lexicon**, it is possible
 1840 to automatically expand the lexicon by looking for words that frequently co-occur with
 1841 words in the seed set (Hatzivassiloglou and McKeown, 1997; Qiu et al., 2011).

1842 4.2 Word sense disambiguation

1843 Consider the the following headlines:

- 1844 (4.8) Iraqi head seeks arms
- 1845 (4.9) Prostitutes appeal to Pope
- 1846 (4.10) Drunk gets nine years in violin case²

1847 These headlines are ambiguous because they contain words that have multiple mean-
 1848 ings, or **senses**. Word sense disambiguation is the problem of identifying the intended
 1849 sense of each word token in a document. Word sense disambiguation is part of a larger
 1850 field of research called **lexical semantics**, which is concerned with meanings of the words.

1851 At a basic level, the problem of word sense disambiguation is to identify the correct
 1852 sense for each word token in a document. Part-of-speech ambiguity (e.g., noun versus
 1853 verb) is usually considered to be a different problem, to be solved at an earlier stage.
 1854 From a linguistic perspective, senses are not properties of words, but of **lemmas**, which
 1855 are canonical forms that stand in for a set of inflected words. For example, *arm*/N is a
 1856 lemma that includes the inflected form *arms*/N — the /N indicates that it we are refer-
 1857 ring to the noun, and not its **homonym** *arm*/V, which is another lemma that includes
 1858 the inflected verbs (*arm*/V, *arms*/V, *armed*/V, *arming*/V). Therefore, word sense disam-
 1859 biguation requires first identifying the correct part-of-speech and lemma for each token,

²These examples, and many more, can be found at <http://www.ling.upenn.edu/~beatrice/humor/headlines.html>

1860 and then choosing the correct sense from the inventory associated with the corresponding
 1861 lemma.³ (Part-of-speech tagging is discussed in § 8.1.)

1862 **4.2.1 How many word senses?**

1863 Words sometimes have many more than two senses, as exemplified by the word *serve*:

- 1864 • [FUNCTION]: *The tree stump served as a table*
- 1865 • [CONTRIBUTE TO]: *His evasive replies only served to heighten suspicion*
- 1866 • [PROVIDE]: *We serve only the rawest fish*
- 1867 • [ENLIST]: *She served in an elite combat unit*
- 1868 • [JAIL]: *He served six years for a crime he didn't commit*
- 1869 • [LEGAL]: *They were served with subpoenas*⁴

1870 These sense distinctions are annotated in **WordNet** (<http://wordnet.princeton.edu>), a lexical semantic database for English. WordNet consists of roughly 100,000 **synsets**,
 1871 which are groups of lemmas (or phrases) that are synonymous. An example synset is
 1872 {*chump*¹, *fool*², *sucker*¹, *mark*⁹}, where the superscripts index the sense of each lemma that
 1873 is included in the synset: for example, there are at least eight other senses of *mark* that
 1874 have different meanings, and are not part of this synset. A lemma is **polysemous** if it
 1875 participates in multiple synsets.

1877 WordNet defines the scope of the word sense disambiguation problem, and, more
 1878 generally, formalizes lexical semantic knowledge of English. (WordNets have been cre-
 1879 ated for a few dozen other languages, at varying levels of detail.) Some have argued
 1880 that WordNet's sense granularity is too fine (Ide and Wilks, 2006); more fundamentally,
 1881 the premise that word senses can be differentiated in a task-neutral way has been criti-
 1882 cized as linguistically naïve (Kilgarriff, 1997). One way of testing this question is to ask
 1883 whether people tend to agree on the appropriate sense for example sentences: accord-
 1884 ing to Mihalcea et al. (2004), people agree on roughly 70% of examples using WordNet
 1885 senses; far better than chance, but less than agreement on other tasks, such as sentiment
 1886 annotation (Wilson et al., 2005).

1887 ***Other lexical semantic relations** Besides **synonymy**, WordNet also describes many
 1888 other lexical semantic relationships, including:

- 1889 • **antonymy**: *x* means the opposite of *y*, e.g. FRIEND-ENEMY;

³Navigli (2009) provides a survey of approaches for word-sense disambiguation.

⁴Several of the examples are adapted from WordNet (Fellbaum, 2010).

- **hyponymy:** x is a special case of y , e.g. RED-COLOR; the inverse relationship is **hyperonymy**;
- **meronymy:** x is a part of y , e.g., WHEEL-BICYCLE; the inverse relationship is **holonymy**.

Classification of these relations can be performed by searching for characteristic patterns between pairs of words, e.g., X , *such as* Y , which signals hyponymy (Hearst, 1992), or X *but* Y , which signals antonymy (Hatzivassiloglou and McKeown, 1997). Another approach is to analyze each term's **distributional statistics** (the frequency of its neighboring words). Such approaches are described in detail in chapter 14.

4.2.2 Word sense disambiguation as classification

How can we tell living *plants* from manufacturing *plants*? The context is often critical:

- (4.11) Town officials are hoping to attract new manufacturing plants through weakened environmental regulations.
- (4.12) The endangered plants play an important role in the local ecosystem.

It is possible to build a feature vector using the bag-of-words representation, by treating each context as a pseudo-document. The feature function is then,

$$\begin{aligned} f((\text{plant}, \text{The endangered plants play an ...}), y) = \\ \{(the, y) : 1, (\text{endangered}, y) : 1, (\text{play}, y) : 1, (\text{an}, y) : 1, \dots\} \end{aligned}$$

As in document classification, many of these features are irrelevant, but a few are very strong predictors. In this example, the context word *endangered* is a strong signal that the intended sense is biology rather than manufacturing. We would therefore expect a learning algorithm to assign high weight to (*endangered*, BIOLOGY), and low weight to (*endangered*, MANUFACTURING).⁵

It may also be helpful to go beyond the bag-of-words: for example, one might encode the position of each context word with respect to the target, e.g.,

$$\begin{aligned} f((\text{bank}, I \text{ went to the bank to deposit my paycheck}), y) = \\ \{(i - 3, \text{went}, y) : 1, (i + 2, \text{deposit}, y) : 1, (i + 4, \text{paycheck}, y) : 1\} \end{aligned}$$

These are called **collocation features**, and they give more information about the specific role played by each context word. This idea can be taken further by incorporating additional syntactic information about the grammatical role played by each context feature, such as the **dependency path** (see chapter 11).

⁵The context bag-of-words can be also used to perform word-sense disambiguation without machine learning: the Lesk (1986) algorithm selects the word sense whose dictionary definition best overlaps the local context.

Using such features, a classifier can be trained from labeled data. A **semantic concordance** is a corpus in which each open-class word (nouns, verbs, adjectives, and adverbs) is tagged with its word sense from the target dictionary or thesaurus. SemCor is a semantic concordance built from 234K tokens of the Brown corpus (Francis and Kucera, 1982), annotated as part of the WordNet project (Fellbaum, 2010). SemCor annotations look like this:

(4.13) As of Sunday¹_N night¹_N there was⁴_V no word²_N ...,

with the superscripts indicating the annotated sense of each polysemous word, and the subscripts indicating the part-of-speech.

As always, supervised classification is only possible if enough labeled examples can be accumulated. This is difficult in word sense disambiguation, because each polysemous lemma requires its own training set: having a good classifier for the senses of *serve* is no help towards disambiguating *plant*. For this reason, **unsupervised** and **semisupervised** methods are particularly important for word sense disambiguation (e.g., Yarowsky, 1995). These methods will be discussed in chapter 5. Unsupervised methods typically lean on the heuristic of “one sense per discourse”, which means that a lemma will usually have a single, consistent sense throughout any given document (Gale et al., 1992). Based on this heuristic, we can propagate information from high-confidence instances to lower-confidence instances in the same document (Yarowsky, 1995).

4.3 Design decisions for text classification

Text classification involves a number of design decisions. In some cases, the design decision is clear from the mathematics: if you are using regularization, then a regularization weight λ must be chosen. Other decisions are more subtle, arising only in the low level “plumbing” code that ingests and processes the raw data. Such decision can be surprisingly consequential for classification accuracy.

4.3.1 What is a word?

The bag-of-words representation presupposes that extracting a vector of word counts from text is unambiguous. But text documents are generally represented as sequences of characters (in an encoding such as ascii or unicode), and the conversion to bag-of-words presupposes a definition of the “words” that are to be counted.

4.3.1.1 Tokenization

The first subtask for constructing a bag-of-words vector is **tokenization**: converting the text from a sequence of characters to a sequence of **word tokens**. A simple approach is

Whitespace	Isn't Ahab, Ahab? ;)
Treebank	Is n't Ahab , Ahab ? ;)
Tweet	Isn't Ahab , Ahab ? ;)
TokTok (Dehdari, 2014)	Isn ' t Ahab , Ahab ? ;)

Figure 4.1: The output of four `nltk` tokenizers, applied to the string *Isn't Ahab, Ahab? ;)*

1945 to define a subset of characters as whitespace, and then split the text on these tokens.
 1946 However, whitespace-based tokenization is not ideal: we may want to split conjunctions
 1947 like *isn't* and hyphenated phrases like *prize-winning* and *half-asleep*, and we likely want
 1948 to separate words from commas and periods that immediately follow them. At the same
 1949 time, it would be better not to split abbreviations like *U.S.* and *Ph.D.* In languages with
 1950 Roman scripts, tokenization is typically performed using regular expressions, with mod-
 1951 ules designed to handle each of these cases. For example, the `nltk` package includes a
 1952 number of tokenizers (Loper and Bird, 2002); the outputs of four of the better-known tok-
 1953 enizers are shown in Figure 4.1. Social media researchers have found that emoticons and
 1954 other forms of orthographic variation pose new challenges for tokenization, leading to the
 1955 development of special purpose tokenizers to handle these phenomena (O'Connor et al.,
 1956 2010).

1957 Tokenization is a language-specific problem, and each language poses unique chal-
 1958 lenges. For example, Chinese does not include spaces between words, nor any other
 1959 consistent orthographic markers of word boundaries. A “greedy” approach is to scan the
 1960 input for character substrings that are in a predefined lexicon. However, Xue et al. (2003)
 1961 notes that this can be ambiguous, since many character sequences could be segmented in
 1962 multiple ways. Instead, he trains a classifier to determine whether each Chinese character,
 1963 or *hanzi*, is a word boundary. More advanced sequence labeling methods for word seg-
 1964 mentation are discussed in § 8.4. Similar problems can occur in languages with alphabetic
 1965 scripts, such as German, which does not include whitespace in compound nouns, yield-
 1966 ing examples such as *Freundschaftsbezeugungen* (demonstration of friendship) and *Dilett-*
 1967 *tantenaufdringlichkeiten* (the importunities of dilettantes). As Twain (1997) argues, “*These*
 1968 *things are not words, they are alphabetic processions.*” Social media raises similar problems
 1969 for English and other languages, with hashtags such as *#TrueLoveInFourWords* requiring
 1970 decomposition for analysis (Brun and Roux, 2014).

1971 4.3.1.2 Normalization

1972 After splitting the text into tokens, the next question is which tokens are really distinct.
 1973 Is it necessary to distinguish *great*, *Great*, and *GREAT*? Sentence-initial capitalization may
 1974 be irrelevant to the classification task. Going further, the complete elimination of case
 1975 distinctions will result in a smaller vocabulary, and thus smaller feature vectors. However,

Original	The	Williams	sisters	are	leaving	this	tennis	centre
Porter stemmer	the	william	sister	are	leav	thi	tenni	centr
Lancaster stemmer	the	william	sist	ar	leav	thi	ten	cent
WordNet lemmatizer	The	Williams	sister	are	leaving	this	tennis	centre

Figure 4.2: Sample outputs of the Porter (1980) and Lancaster (Paice, 1990) stemmers, and the WordNet lemmatizer

1976 case distinctions might be relevant in some situations: for example, *apple* is a delicious
 1977 pie filling, while *Apple* is a company that specializes in proprietary dongles and power
 1978 adapters.

1979 For Roman script, case conversion can be performed using unicode string libraries.
 1980 Many scripts do not have case distinctions (e.g., the Devanagari script used for South
 1981 Asian languages, the Thai alphabet, and Japanese kana), and case conversion for all scripts
 1982 may not be available in every programming environment. (Unicode support is an im-
 1983 portant distinction between Python’s versions 2 and 3, and is a good reason for mi-
 1984 grating to Python 3 if you have not already done so. Compare the output of the code
 1985 "\à l'hôtel".upper() in the two language versions.)⁶

1986 Case conversion is a type of **normalization**, which refers to string transformations that
 1987 remove distinctions that are irrelevant to downstream applications (Sproat et al., 2001).
 1988 Other normalizations include the standardization of numbers (e.g., 1,000 to 1000) and
 1989 dates (e.g., August 11, 2015 to 2015/11/08). Depending on the application, it may even be
 1990 worthwhile to convert all numbers and dates to special tokens, !NUM and !DATE. In social
 1991 media, there are additional orthographic phenomena that may be normalized, such as ex-
 1992 pressive lengthening, e.g., *coooooool* (Aw et al., 2006; Yang and Eisenstein, 2013). Similarly,
 1993 historical texts feature spelling variations that may need to be normalized to a contempo-
 1994 rary standard form (Baron and Rayson, 2008).

1995 A more extreme form of normalization is to eliminate **inflectional affixes**, such as the
 1996 -ed and -s suffixes in English. On this view, *bike*, *bikes*, *biking*, and *biked* all refer to the
 1997 same underlying concept, so they should be grouped into a single feature. A **stemmer** is
 1998 a program for eliminating affixes, usually by applying a series of regular expression sub-
 1999 stitutions. Character-based stemming algorithms are necessarily approximate, as shown
 2000 in Figure 4.2: the Lancaster stemmer incorrectly identifies -ers as an inflectional suffix of
 2001 *sisters* (by analogy to *fix/fixer*s), and both stemmers incorrectly identify -s as a suffix of *this*
 2002 and *Williams*. Fortunately, even inaccurate stemming can improve bag-of-words classifi-
 2003 cation models, by merging related strings and thereby reducing the vocabulary size.

2004 Accurately handling irregular orthography requires word-specific rules. **Lemmatizers**

⁶[todo: I want to make this a footnote, but can't figure out how.]

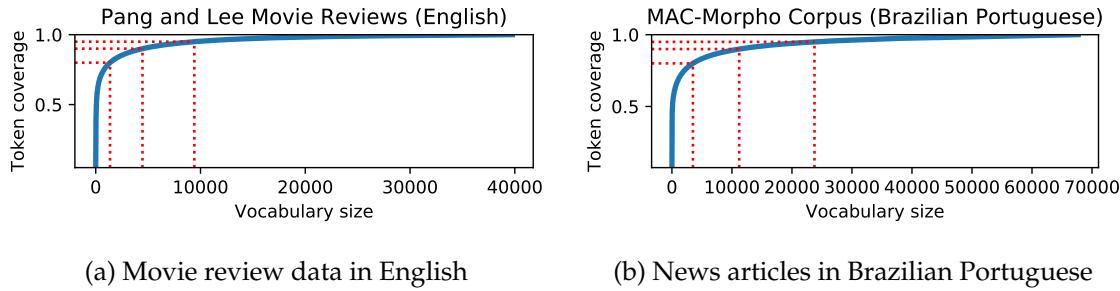


Figure 4.3: Tradeoff between token coverage (y-axis) and vocabulary size, on the `nltk` movie review dataset, after sorting the vocabulary by decreasing frequency. The red dashed lines indicate 80%, 90%, and 95% coverage.

2005 are systems that identify the underlying lemma of a given wordform. They must avoid the
 2006 over-generalization errors of the stemmers in Figure 4.2, and also handle more complex
 2007 transformations, such as *geese*→*goose*. The output of the WordNet lemmatizer is shown in
 2008 the final line of Figure 4.2. Both stemming and lemmatization are language-specific: an
 2009 English stemmer or lemmatizer is of little use on a text written in another language. The
 2010 discipline of **morphology** relates to the study of word-internal structure, and is described
 2011 in more detail in § 9.1.2.

2012 The value of normalization depends on the data and the task. Normalization re-
 2013 duces the size of the feature space, which can help in generalization. However, there
 2014 is always the risk of merging away linguistically meaningful distinctions. In supervised
 2015 machine learning, regularization and smoothing can play a similar role to normalization
 2016 — preventing the learner from overfitting to rare features — while avoiding the language-
 2017 specific engineering required for accurate normalization. In unsupervised scenarios, such
 2018 as content-based information retrieval (Manning et al., 2008) and topic modeling (Blei
 2019 et al., 2003), normalization is more critical.

2020 4.3.2 How many words?

2021 Limiting the size of the feature vector reduces the memory footprint of the resulting mod-
 2022 els, and increases the speed of prediction. Normalization can help to play this role, but
 2023 a more direct approach is simply to limit the vocabulary to the N most frequent words
 2024 in the dataset. For example, in the `movie-reviews` dataset provided with `nltk` (orig-
 2025 inally from Pang et al., 2002), there are 39,768 word types, and 1.58M tokens. As shown
 2026 in Figure 4.3a, the most frequent 4000 word types cover 90% of all tokens, offering an
 2027 order-of-magnitude reduction in the model size. Such ratios are language-specific: in for
 2028 example, in the Brazilian Portuguese Mac-Morpho corpus (Aluísio et al., 2003), attain-
 2029 ing 90% coverage requires more than 10000 word types (Figure 4.3b). This reflects the

2030 morphological complexity of Portuguese, which includes many more inflectional suffixes
 2031 than English.

2032 Eliminating rare words is not always advantageous for classification performance: for
 2033 example, names, which are typically rare, play a large role in distinguishing topics of news
 2034 articles. Another way to reduce the size of the feature space is to eliminate **stopwords** such
 2035 as *the*, *to*, and *and*, which may seem to play little role in expressing the topic, sentiment,
 2036 or stance. This is typically done by creating a **stoplist** (e.g., `nltk.corpus.stopwords`),
 2037 and then ignoring all terms that match the list. However, corpus linguists and social psy-
 2038 chologists have shown that seemingly inconsequential words can offer surprising insights
 2039 about the author or nature of the text (Biber, 1991; Chung and Pennebaker, 2007). Further-
 2040 more, high-frequency words are unlikely to cause overfitting in discriminative classifiers.
 2041 As with normalization, stopword filtering is more important for unsupervised problems,
 2042 such as term-based document retrieval.

2043 Another alternative for controlling model size is **feature hashing** (Weinberger et al.,
 2044 2009). Each feature is assigned an index using a hash function. If a hash function that
 2045 permits collisions is chosen (typically by taking the hash output modulo some integer),
 2046 then the model can be made arbitrarily small, as multiple features share a single weight.
 2047 Because most features are rare, accuracy is surprisingly robust to such collisions (Ganchev
 2048 and Dredze, 2008).

2049 4.3.3 Count or binary?

2050 Finally, we may consider whether we want our feature vector to include the *count* of each
 2051 word, or its *presence*. This gets at a subtle limitation of linear classification: it worse to
 2052 have two *failures* than one, but is it really twice as bad? Motivated by this intuition, Pang
 2053 et al. (2002) use binary indicators of presence or absence in the feature vector: $f_j(x, y) \in$
 2054 $\{0, 1\}$. They find that classifiers trained on these binary vectors tend to outperform feature
 2055 vectors based on word counts. One explanation is that words tend to appear in clumps:
 2056 if a word has appeared once in a document, it is likely to appear again (Church, 2000).
 2057 These subsequent appearances can be attributed to this tendency towards repetition, and
 2058 thus provide little additional information about the class label of the document.

2059 4.4 Evaluating classifiers

2060 In any supervised machine learning application, it is critical to reserve a held-out test set.
 2061 This data should be used for only one purpose: to evaluate the overall accuracy of a single
 2062 classifier. Using this data more than once would cause the estimated accuracy to be overly
 2063 optimistic, because the classifier would be customized to this data, and would not perform
 2064 as well as on unseen data in the future. It is usually necessary to set hyperparameters or

2065 perform feature selection, so you may need to construct a **tuning** or **development set** for
 2066 this purpose, as discussed in § 2.1.5.

2067 There are a number of ways to evaluate classifier performance. The simplest is **accuracy**:
 2068 the number of correct predictions, divided by the total number of instances,

$$\text{acc}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_i^N \delta(y^{(i)} = \hat{y}). \quad [4.4]$$

2069 Exams are usually graded by accuracy. Why are other metrics necessary? The main
 2070 reason is **class imbalance**. Suppose you are building a classifier to detect whether an
 2071 electronic health record (EHR) describes symptoms of a rare disease, which appears in
 2072 only 1% of all documents in the dataset. A classifier that reports $\hat{y} = \text{NEGATIVE}$ for
 2073 all documents would achieve 99% accuracy, but would be practically useless. We need
 2074 metrics that are capable of detecting the classifier's ability to discriminate between classes,
 2075 even when the distribution is skewed.

2076 One solution is to build a **balanced test set**, in which each possible label is equally rep-
 2077 resented. But in the EHR example, this would mean throwing away 98% of the original
 2078 dataset! Furthermore, the detection threshold itself might be a design consideration: in
 2079 health-related applications, we might prefer a very sensitive classifier, which returned a
 2080 positive prediction if there is even a small chance that $y^{(i)} = \text{POSITIVE}$. In other applica-
 2081 tions, a positive result might trigger a costly action, so we would prefer a classifier that
 2082 only makes positive predictions when absolutely certain. We need additional metrics to
 2083 capture these characteristics.

2084 4.4.1 Precision, recall, and F-MEASURE

2085 For any label (e.g., positive for presence of symptoms of a disease), there are two possible
 2086 errors:

- 2087 • **False positive**: the system incorrectly predicts the label.
- 2088 • **False negative**: the system incorrectly fails to predict the label.

2089 Similarly, for any label, there are two ways to be correct:

- 2090 • **True positive**: the system correctly predicts the label.
- 2091 • **True negative**: the system correctly predicts that the label does not apply to this
 2092 instance.

Classifiers that make a lot of false positives are too sensitive; classifiers that make a
 lot of false negatives are not sensitive enough. These two conditions are captured by the

metrics of **recall** and **precision**:

$$\text{RECALL}(\mathbf{y}, \hat{\mathbf{y}}, k) = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad [4.5]$$

$$\text{PRECISION}(\mathbf{y}, \hat{\mathbf{y}}, k) = \frac{\text{TP}}{\text{TP} + \text{FP}}. \quad [4.6]$$

2093 Recall and precision are both conditional likelihoods of a correct prediction, which is why
 2094 their numerators are the same. Recall is conditioned on k being the correct label, $y^{(i)} = k$,
 2095 so the denominator sums over true positive and false negatives. Precision is conditioned
 2096 on k being the prediction, so the denominator sums over true positives and false positives.
 2097 Note that true negatives are not considered in either statistic. The classifier that labels
 2098 every document as “negative” would achieve zero recall; precision would be $\frac{0}{0}$.

2099 Recall and precision are complementary. A high-recall classifier is preferred when
 2100 false negatives are cheaper than false positives: for example, in a preliminary screening
 2101 for symptoms of a disease, the cost of a false positive might be an additional test, while a
 2102 false negative would result in the disease going untreated. Conversely, a high-precision
 2103 classifier is preferred when false positives are more expensive: for example, in spam de-
 2104 tection, a false negative is a relatively minor inconvenience, while a false positive might
 2105 mean that an important message goes unread.

The **F -MEASURE** combines recall and precision into a single metric, using the harmonic mean:

$$F\text{-MEASURE}(\mathbf{y}, \hat{\mathbf{y}}, k) = \frac{2rp}{r + p}, \quad [4.7]$$

2106 where r is recall and p is precision.⁷

Evaluating multi-class classification Recall, precision, and **F -MEASURE** are defined with respect to a specific label k . When there are multiple labels of interest (e.g., in word sense disambiguation or emotion classification), it is necessary to combine the **F -MEASURE** across each class. **Macro F -MEASURE** is the average **F -MEASURE** across several classes,

$$\text{Macro-}F(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} F\text{-MEASURE}(\mathbf{y}, \hat{\mathbf{y}}, k) \quad [4.8]$$

2107 In multi-class problems with unbalanced class distributions, the macro **F -MEASURE** is a
 2108 balanced measure of how well the classifier recognizes each class. In **micro F -MEASURE**,
 2109 we compute true positives, false positives, and false negatives for each class, and then add
 2110 them up to compute a single recall, precision, and **F -MEASURE**. This metric is balanced
 2111 across instances rather than classes, so it weights each class in proportion to its frequency
 2112 — unlike macro **F -MEASURE**, which weights each class equally.

⁷ F -MEASURE is sometimes called F_1 , and generalizes to $F_\beta = \frac{(1+\beta^2)rp}{\beta^2p+r}$. The β parameter can be tuned to emphasize recall or precision.

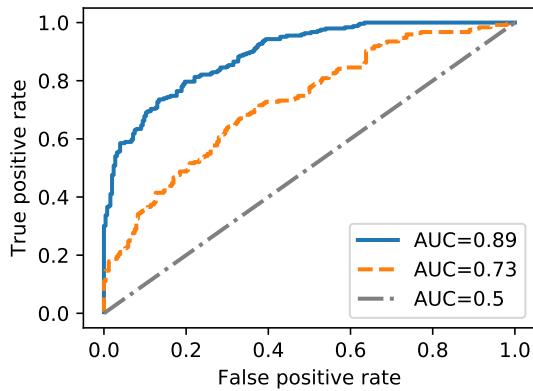


Figure 4.4: ROC curves for three classifiers of varying discriminative power, measured by AUC (area under the curve)

2113 4.4.2 Threshold-free metrics

2114 In binary classification problems, it is possible to trade off between recall and precision by
 2115 adding a constant “threshold” to the output of the scoring function. This makes it possible
 2116 to trace out a curve, where each point indicates the performance at a single threshold. In
 2117 the **receiver operating characteristic (ROC)** curve,⁸ the *x*-axis indicates the **false positive**
 2118 **rate**, $\frac{FP}{FP+TN}$, and the *y*-axis indicates the recall, or **true positive rate**. A perfect classifier
 2119 attains perfect recall without any false positives, tracing a “curve” from the origin (0,0) to
 2120 the upper left corner (0,1), and then to (1,1). In expectation, a non-discriminative classifier
 2121 traces a diagonal line from the origin (0,0) to the upper right corner (1,1). Real classifiers
 2122 tend to fall between these two extremes. Examples are shown in Figure 4.4.

2123 The ROC curve can be summarized in a single number by taking its integral, the **area**
 2124 **under the curve (AUC)**. The AUC can be interpreted as the probability that a randomly-
 2125 selected positive example will be assigned a higher score by the classifier than a randomly-
 2126 selected negative example. A perfect classifier has AUC = 1 (all positive examples score
 2127 higher than all negative examples); a non-discriminative classifier has AUC = 0.5 (given
 2128 a randomly selected positive and negative example, either could score higher with equal
 2129 probability); a perfectly wrong classifier would have AUC = 0 (all negative examples score
 2130 higher than all positive examples). One advantage of AUC in comparison to *F*-MEASURE
 2131 is that the baseline rate of 0.5 does not depend on the label distribution.

⁸The name “receiver operator characteristic” comes from the metric’s origin in signal processing applications (Peterson et al., 1954). Other threshold-free metrics include **precision-recall curves**, **precision-at-*k***, and **balanced *F*-MEASURE**; see Manning et al. (2008) for more details.

2132 **4.4.3 Classifier comparison and statistical significance**

2133 Natural language processing research and engineering often involves comparing different
 2134 classification techniques. In some cases, the comparison is between algorithms, such as
 2135 logistic regression versus averaged perceptron, or L_2 regularization versus L_1 . In other
 2136 cases, the comparison is between feature sets, such as the bag-of-words versus positional
 2137 bag-of-words (see § 4.2.2). **Ablation testing** involves systematically removing (ablating)
 2138 various aspects of the classifier, such as feature groups, and testing the **null hypothesis**
 2139 that the ablated classifier is as good as the full model.

2140 A full treatment of hypothesis testing is beyond the scope of this text, but this section
 2141 contains a brief summary of the techniques necessary to compare classifiers. The main
 2142 aim of hypothesis testing is to determine whether the difference between two statistics
 2143 — for example, the accuracies of two classifiers — is likely to arise by chance. We will
 2144 be concerned with chance fluctuations that arise due to the finite size of the test set.⁹ An
 2145 improvement of 10% on a test set with ten instances may reflect a random fluctuation that
 2146 makes the test set more favorable to classifier c_1 than c_2 ; on another test set with a different
 2147 ten instances, we might find that c_2 does better than c_1 . But if we observe the same 10%
 2148 improvement on a test set with 1000 instances, this is highly unlikely to be explained
 2149 by chance. Such a finding is said to be **statistically significant** at a level p , which is the
 2150 probability of observing an effect of equal or greater magnitude when the null hypothesis
 2151 is true. The notation $p < .05$ indicates that the likelihood of an equal or greater effect is
 2152 less than 5%, assuming the null hypothesis is true.¹⁰

2153 **4.4.3.1 The binomial test**

2154 The statistical significance of a difference in accuracy can be evaluated using classical tests,
 2155 such as the **binomial test**.¹¹ Suppose that classifiers c_1 and c_2 disagree on N instances in a
 2156 test set with binary labels, and that c_1 is correct on k of those instances. Under the null hy-
 2157 pothesis that the classifiers are equally accurate, we would expect k/N to be roughly equal
 2158 to 1/2, and as N increases, k/N should be increasingly close to this expected value. These
 2159 properties are captured by the **binomial distribution**, which is a probability over counts

⁹Other sources of variance include the initialization of non-convex classifiers such as neural networks, and the ordering of instances in online learning such as stochastic gradient descent and perceptron.

¹⁰Statistical hypothesis testing is useful only to the extent that the existing test set is representative of the instances that will be encountered in the future. If, for example, the test set is constructed from news documents, no hypothesis test can predict which classifier will perform best on documents from another domain, such as electronic health records.

¹¹A well-known alternative to the binomial test is **McNemar's test**, which computes a **test statistic** based on the number of examples that are correctly classified by one system and incorrectly classified by the other. The null hypothesis distribution for this test statistic is known to be drawn from a chi-squared distribution with a single degree of freedom, so a p -value can be computed from the cumulative density function of this distribution (Dietterich, 1998). Both tests give similar results in most circumstances, but the binomial test is easier to understand from first principles.

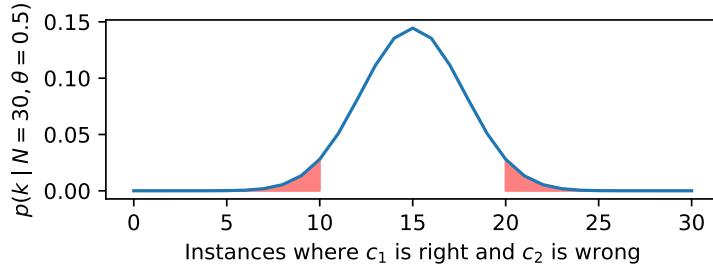


Figure 4.5: Probability mass function for the binomial distribution. The pink highlighted areas represent the cumulative probability for a significance test on an observation of $k = 10$ and $N = 30$.

of binary random variables. We write $k \sim \text{Binom}(\theta, N)$ to indicate that k is drawn from a binomial distribution, with parameter N indicating the number of random “draws”, and θ indicating the probability of “success” on each draw. Each draw is an example on which the two classifiers disagree, and a “success” is a case in which c_1 is right and c_2 is wrong. (The label space is assumed to be binary, so if the classifiers disagree, exactly one of them is correct. The test can be generalized to multi-class classification by focusing on the examples in which exactly one classifier is correct.)

The probability mass function (PMF) of the binomial distribution is,

$$p_{\text{Binom}}(k; N, \theta) = \binom{N}{k} \theta^k (1 - \theta)^{N-k}, \quad [4.9]$$

with θ^k representing the probability of the k successes, $(1 - \theta)^{N-k}$ representing the probability of the $N - k$ unsuccessful draws. The expression $\binom{N}{k} = \frac{N!}{k!(N-k)!}$ is a binomial coefficient, representing the number of possible orderings of events; this ensures that the distribution sums to one over all $k \in \{0, 1, 2, \dots, N\}$.

Under the null hypothesis, when the classifiers disagree, each classifier is equally likely to be right, so $\theta = \frac{1}{2}$. Now suppose that among N disagreements, c_1 is correct $k < \frac{N}{2}$ times. The probability of c_1 being correct k or fewer times is the **one-tailed p-value**, because it is computed from the area under the binomial probability mass function from 0 to k , as shown in the left tail of Figure 4.5. This **cumulative probability** is computed as a sum over all values $i \leq k$,

$$\Pr_{\text{Binom}} \left(\text{count}(\hat{y}_2^{(i)} = y^{(i)} \neq \hat{y}_1^{(i)}) \leq k; N, \theta = \frac{1}{2} \right) = \sum_{i=0}^k p_{\text{Binom}} \left(i; N, \theta = \frac{1}{2} \right). \quad [4.10]$$

The one-tailed p-value applies only to the asymmetric null hypothesis that c_1 is at least as accurate as c_2 . To test the **two-tailed** null hypothesis that c_1 and c_2 are equally accu-

Algorithm 7 Bootstrap sampling for classifier evaluation. The original test set is $\{\mathbf{x}^{(1:N)}, \mathbf{y}^{(1:N)}\}$, the metric is $\delta(\cdot)$, and the number of samples is M .

```

procedure BOOTSTRAP-SAMPLE( $\mathbf{x}^{(1:N)}, \mathbf{y}^{(1:N)}, \delta(\cdot), M$ )
  for  $t \in \{1, 2, \dots, M\}$  do
    for  $i \in \{1, 2, \dots, N\}$  do
       $j \sim \text{UniformInteger}(1, N)$ 
       $\tilde{\mathbf{x}}^{(i)} \leftarrow \mathbf{x}^{(j)}$ 
       $\tilde{\mathbf{y}}^{(i)} \leftarrow \mathbf{y}^{(j)}$ 
       $d^{(t)} \leftarrow \delta(\tilde{\mathbf{x}}^{(1:N)}, \tilde{\mathbf{y}}^{(1:N)})$ 
  return  $\{d^{(t)}\}_{t=1}^M$ 
```

2174 rate, we would take the sum of one-tailed p -values, where the second term is computed
 2175 from the right tail of Figure 4.5. The binomial distribution is symmetric, so this can be
 2176 computed by simply doubling the one-tailed p -value.

2177 Two-tailed tests are more stringent, but they are necessary in cases in which there is
 2178 no prior intuition about whether c_1 or c_2 is better. For example, in comparing logistic
 2179 regression versus averaged perceptron, a two-tailed test is appropriate. In an ablation
 2180 test, c_2 may contain a superset of the features available to c_1 . If the additional features are
 2181 thought to be likely to improve performance, then a one-tailed test would be appropriate,
 2182 if chosen in advance. However, such a test can only prove that c_2 is more accurate than
 2183 c_1 , and not the reverse.

2184 **4.4.3.2 *Randomized testing**

2185 The binomial test is appropriate for accuracy, but not for more complex metrics such as
 2186 F -MEASURE. To compute statistical significance for arbitrary metrics, we can apply ran-
 2187 domization. Specifically, draw a set of M **bootstrap samples** (Efron and Tibshirani, 1993),
 2188 by resampling instances from the original test set with replacement. Each bootstrap sam-
 2189 ple is itself a test set of size N . Some instances from the original test set will not appear
 2190 in any given bootstrap sample, while others will appear multiple times; but overall, the
 2191 sample will be drawn from the same distribution as the original test set. We can then com-
 2192 pute any desired evaluation on each bootstrap sample, which gives a distribution over the
 2193 value of the metric. Algorithm 7 shows how to perform this computation.

2194 To compare the F -MEASURE of two classifiers c_1 and c_2 , we set the function $\delta(\cdot)$ to
 2195 compute the difference in F -MEASURE on the bootstrap sample. If the difference is less
 2196 than or equal to zero in at least 5% of the samples, then we cannot reject the one-tailed
 2197 null hypothesis that c_2 is at least as good as c_1 (Berg-Kirkpatrick et al., 2012). We may
 2198 also be interested in the 95% **confidence interval** around a metric of interest, such as
 2199 the F -MEASURE of a single classifier. This can be computed by sorting the output of

2200 Algorithm 7, and then setting the top and bottom of the 95% confidence interval to the
 2201 values at the 2.5% and 97.5% percentiles of the sorted outputs. Alternatively, you can fit
 2202 a normal distribution to the set of differences across bootstrap samples, and compute a
 2203 Gaussian confidence interval from the mean and variance.

2204 As the number of bootstrap samples goes to infinity, $M \rightarrow \infty$, the bootstrap estimate
 2205 is increasingly accurate. A typical choice for M is 10^4 or 10^5 ; larger numbers of samples
 2206 are necessary for smaller p -values. One way to validate your choice of M is to run the test
 2207 multiple times, and ensure that the p -values are similar; if not, increase M by an order of
 2208 magnitude. This is a heuristic measure of the **variance** of the test, which can decrease
 2209 with the square root \sqrt{M} (Robert and Casella, 2013).

2210 4.4.4 *Multiple comparisons

2211 Sometimes it is necessary to perform multiple hypothesis tests, such as when compar-
 2212 ing the performance of several classifiers on multiple datasets. Suppose you have five
 2213 datasets, and you compare four versions of your classifier against a baseline system, for a
 2214 total of 20 comparisons. Even if none of your classifiers is better than the baseline, there
 2215 will be some chance variation in the results, and in expectation you will get one statis-
 2216 tically significant improvement at $p = 0.05 = \frac{1}{20}$. It is therefore necessary to adjust the
 2217 p -values when reporting the results of multiple comparisons.

2218 One approach is to require a threshold of $\frac{\alpha}{m}$ to report a p value of $p < \alpha$ when per-
 2219 forming m tests. This is known as the **Bonferroni correction**, and it limits the overall
 2220 probability of incorrectly rejecting the null hypothesis at α . Another approach is to bound
 2221 the **false discovery rate** (FDR), which is the fraction of null hypothesis rejections that are
 2222 incorrect. Benjamini and Hochberg (1995) propose a p -value correction that bounds the
 2223 fraction of false discoveries at α : sort the p -values of each individual test in ascending
 2224 order, and set the significance threshold equal to largest k such that $p_k \leq \frac{k}{m}\alpha$. If $k > 1$, the
 2225 FDR adjustment is more permissive than the Bonferroni correction.

2226 4.5 Building datasets

2227 Sometimes, if you want to build a classifier, you must first build a dataset of your own.
 2228 This includes selecting a set of documents or instances to annotate, and then performing
 2229 the annotations. The scope of the dataset may be determined by the application: if you
 2230 want to build a system to classify electronic health records, then you must work with a
 2231 corpus of records of the type that your classifier will encounter when deployed. In other
 2232 cases, the goal is to build a system that will work across a broad range of documents. In
 2233 this case, it is best to have a *balanced* corpus, with contributions from many styles and
 2234 genres. For example, the Brown corpus draws from texts ranging from government doc-
 2235 uments to romance novels (Francis, 1964), and the Google Web Treebank includes an-

2236 notations for five “domains” of web documents: question answers, emails, newsgroups,
2237 reviews, and blogs (Petrov and McDonald, 2012).

2238 **4.5.1 Metadata as labels**

2239 Annotation is difficult and time-consuming, and most people would rather avoid it. It
2240 is sometimes possible to exploit existing metadata to obtain labels for training a classi-
2241 fier. For example, reviews are often accompanied by a numerical rating, which can be
2242 converted into a classification label (see § 4.1). Similarly, the nationalities of social media
2243 users can be estimated from their profiles (Dredze et al., 2013) or even the time zones of
2244 their posts (Gouws et al., 2011). More ambitiously, we may try to classify the political af-
2245 filiations of social media profiles based on their social network connections to politicians
2246 and major political parties (Rao et al., 2010).

2247 The convenience of quickly constructing large labeled datasets without manual an-
2248 notation is appealing. However this approach relies on the assumption that unlabeled
2249 instances — for which metadata is unavailable — will be similar to labeled instances.
2250 Consider the example of labeling the political affiliation of social media users based on
2251 their network ties to politicians. If a classifier attains high accuracy on such a test set,
2252 is it safe to assume that it accurately predicts the political affiliation of all social media
2253 users? Probably not. Social media users who establish social network ties to politicians
2254 may be more likely to mention politics in the text of their messages, as compared to the
2255 average user, for whom no political metadata is available. If so, the accuracy on a test set
2256 constructed from social network metadata would give an overly optimistic picture of the
2257 method’s true performance on unlabeled data.

2258 **4.5.2 Labeling data**

2259 In many cases, there is no way to get ground truth labels other than manual annotation.
2260 An annotation protocol should satisfy several criteria: the annotations should be *expressive*
2261 enough to capture the phenomenon of interest; they should be *replicable*, meaning that
2262 another annotator or team of annotators would produce very similar annotations if given
2263 the same data; and they should be *scalable*, so that they can be produced relatively quickly.
2264 Hovy and Lavid (2010) propose a structured procedure for obtaining annotations that
2265 meet these criteria, which is summarized below.

- 2266 1. **Determine what the annotations are to include.** This is usually based on some
2267 theory of the underlying phenomenon: for example, if the goal is to produce an-
2268 notations about the emotional state of a document’s author, one should start with a
2269 theoretical account of the types or dimensions of emotion (e.g., Mohammad and Tur-
2270 ney, 2013). At this stage, the tradeoff between expressiveness and scalability should

2271 be considered: a full instantiation of the underlying theory might be too costly to
2272 annotate at scale, so reasonable approximations should be considered.

- 2273 2. Optionally, one may **design or select a software tool to support the annotation**
2274 **effort**. Existing general-purpose annotation tools include BRAT (Stenetorp et al.,
2275 2012) and MMAX2 (Müller and Strube, 2006).
- 2276 3. **Formalize the instructions for the annotation task.** To the extent that the instruc-
2277 tions are not explicit, the resulting annotations will depend on the intuitions of the
2278 annotators. These intuitions may not be shared by other annotators, or by the users
2279 of the annotated data. Therefore explicit instructions are critical to ensuring the an-
2280 notations are replicable and usable by other researchers.
- 2281 4. **Perform a pilot annotation** of a small subset of data, with multiple annotators for
2282 each instance. This will give a preliminary assessment of both the replicability and
2283 scalability of the current annotation instructions. Metrics for computing the rate of
2284 agreement are described below. Manual analysis of specific disagreements should
2285 help to clarify the instructions, and may lead to modifications of the annotation task
2286 itself. For example, if two labels are commonly conflated by annotators, it may be
2287 best to merge them.
- 2288 5. **Annotate the data.** After finalizing the annotation protocol and instructions, the
2289 main annotation effort can begin. Some, if not all, of the instances should receive
2290 multiple annotations, so that inter-annotator agreement can be computed. In some
2291 annotation projects, instances receive many annotations, which are then aggregated
2292 into a “consensus” label (e.g., Danescu-Niculescu-Mizil et al., 2013). However, if the
2293 annotations are time-consuming or require significant expertise, it may be preferable
2294 to maximize scalability by obtaining multiple annotations for only a small subset of
2295 examples.
- 2296 6. **Compute and report inter-annotator agreement, and release the data.** In some
2297 cases, the raw text data cannot be released, due to concerns related to copyright or
2298 privacy. In these cases, one solution is to publicly release **stand-off annotations**,
2299 which contain links to document identifiers. The documents themselves can be re-
2300 leased under the terms of a licensing agreement, which can impose conditions on
2301 how the data is used. It is important to think through the potential consequences of
2302 releasing data: people may make personal data publicly available without realizing
2303 that it could be redistributed in a dataset and publicized far beyond their expecta-
2304 tions (boyd and Crawford, 2012).

2305 **4.5.2.1 Measuring inter-annotator agreement**

2306 To measure the replicability of annotations, a standard practice is to compute the extent to
 2307 which annotators agree with each other. If the annotators frequently disagree, this casts
 2308 doubt on either their reliability or on the annotation system itself. For classification, one
 2309 can compute the frequency with which the annotators agree; for rating scales, one can
 2310 compute the average distance between ratings. These raw agreement statistics must then
 2311 be compared with the rate of **chance agreement** — the level of agreement that would be
 2312 obtained between two annotators who ignored the data.

2313 **Cohen's Kappa** is widely used for quantifying the agreement on discrete labeling
 2314 tasks (Cohen, 1960; Carletta, 1996),¹²

$$\kappa = \frac{\text{agreement} - E[\text{agreement}]}{1 - E[\text{agreement}]}. \quad [4.11]$$

2315 The numerator is the difference between the observed agreement and the chance agree-
 2316 ment, and the denominator is the difference between perfect agreement and chance agree-
 2317 ment. Thus, $\kappa = 1$ when the annotators agree in every case, and $\kappa = 0$ when the annota-
 2318 tors agree only as often as would happen by chance. Various heuristic scales have been
 2319 proposed for determining when κ indicates "moderate", "good", or "substantial" agree-
 2320 ment; for reference, Lee and Narayanan (2005) report $\kappa \approx 0.45 - 0.47$ for annotations
 2321 of emotions in spoken dialogues, which they describe as "moderate agreement"; Stolcke
 2322 et al. (2000) report $\kappa = 0.8$ for annotations of **dialogue acts**, which are labels for the pur-
 2323 pose of each turn in a conversation.

2324 When there are two annotators, the expected chance agreement is computed as,

$$E[\text{agreement}] = \sum_k \hat{\Pr}(Y = k)^2, \quad [4.12]$$

2325 where k is a sum over labels, and $\hat{\Pr}(Y = k)$ is the empirical probability of label k across
 2326 all annotations. The formula is derived from the expected number of agreements if the
 2327 annotations were randomly shuffled. Thus, in a binary labeling task, if one label is applied
 2328 to 90% of instances, chance agreement is $.9^2 + .1^2 = .82$.

2329 **4.5.2.2 Crowdsourcing**

2330 Crowdsourcing is often used to rapidly obtain annotations for classification problems.
 2331 For example, **Amazon Mechanical Turk** makes it possible to define "human intelligence
 2332 tasks (hits)", such as labeling data. The researcher sets a price for each set of annotations
 2333 and a list of minimal qualifications for annotators, such as their native language and their

¹² For other types of annotations, Krippendorff's alpha is a popular choice (Hayes and Krippendorff, 2007; Artstein and Poesio, 2008).

2334 satisfaction rate on previous tasks. The use of relatively untrained “crowdworkers” con-
 2335 trasts with earlier annotation efforts, which relied on professional linguists (Marcus et al.,
 2336 1993). However, crowdsourcing has been found to produce reliable annotations for many
 2337 language-related tasks (Snow et al., 2008). Crowdsourcing is part of the broader field of
 2338 **human computation** (Law and Ahn, 2011).

2339 Additional resources

2340 Many of the preprocessing issues discussed in this chapter also arise in information re-
 2341 trieval. See (Manning et al., 2008, chapter 2) for discussion of tokenization and related
 2342 algorithms.

2343 Exercises

2344 1. As noted in § 4.3.3, words tend to appear in clumps, with subsequent occurrences
 2345 of a word being more probable. More concretely, if word j has probability $\phi_{y,j}$
 2346 of appearing in a document with label y , then the probability of two appearances
 2347 ($x_j^{(i)} = 2$) is greater than $\phi_{y,j}^2$.

2348 Suppose you are applying Naïve Bayes to a binary classification. Focus on a word j
 2349 which is more probable under label $y = 1$, so that,

$$\Pr(w = j \mid y = 1) > \Pr(w = j \mid y = 0). \quad [4.13]$$

2350 Now suppose that $x_j^{(i)} > 1$. All else equal, will the classifier overestimate or under-
 2351 estimate the posterior $\Pr(y = 1 \mid x)$?

2352 2. Prove that F-measure is never greater than the arithmetic mean of recall and pre-
 2353 cision, $\frac{r+p}{2}$. Your solution should also show that F-measure is equal to $\frac{r+p}{2}$ iff $r = p$.

2354 3. Given a binary classification problem in which the probability of the “positive” label
 2355 is equal to α , what is the expected *F*-MEASURE of a random classifier which ignores
 2356 the data, and selects $\hat{y} = +1$ with probability $\frac{1}{2}$? (Assume that $p(\hat{y}) \perp p(y)$.) What is
 2357 the expected *F*-MEASURE of a classifier that selects $\hat{y} = +1$ with probability α (also
 2358 independent of $y^{(i)}$)? Depending on α , which random classifier will score better?

2359 4. Suppose that binary classifiers c_1 and c_2 disagree on $N = 30$ cases, and that c_1 is
 2360 correct in $k = 10$ of those cases.

- 2361 • Write a program that uses primitive functions such as `exp` and `factorial` to com-
 2362 pute the **two-tailed** *p*-value — you may use an implementation of the “choose”
 2363 function if one is available. Verify your code against the output of a library for

- 2364 computing the binomial test or the binomial CDF, such as `scipy.stats.binom`
 2365 in Python.
- 2366 • Then use a randomized test to try to obtain the same p -value. In each sample,
 2367 draw from a binomial distribution with $N = 30$ and $\theta = \frac{1}{2}$. Count the fraction
 2368 of samples in which $k \leq 10$. This is the one-tailed p -value; double this to
 2369 compute the two-tailed p -value.
 - 2370 • Try this with varying numbers of bootstrap samples: $M \in \{100, 1000, 5000, 10000\}$.
 2371 For $M = 100$ and $M = 1000$, run the test 10 times, and plot the resulting p -
 2372 values.
 - 2373 • Finally, perform the same tests for $N = 70$ and $k = 25$.
- 2374 5. SemCor 3.0 is a labeled dataset for word sense disambiguation. You can download
 2375 it,¹³ or access it in `nltk.corpora.semcor`.
- 2376 Choose a word that appears at least ten times in SemCor (*find*), and annotate its
 2377 WordNet senses across ten randomly-selected examples, without looking at the ground
 2378 truth. Use online WordNet to understand the definition of each of the senses.¹⁴ Have
 2379 a partner do the same annotations, and compute the raw rate of agreement, expected
 2380 chance rate of agreement, and Cohen's kappa.
- 2381 6. Download the Pang and Lee movie review data, currently available from <http://www.cs.cornell.edu/people/pabo/movie-review-data/>. Hold out a
 2382 randomly-selected 400 reviews as a test set.
- 2383 Download a sentiment lexicon, such as the one currently available from Bing Liu,
 2384 <https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html>. Tokenize
 2385 the data, and classify each document as positive iff it has more positive sentiment
 2386 words than negative sentiment words. Compute the accuracy and *F*-MEASURE on
 2387 detecting positive reviews on the test set, using this lexicon-based classifier.
- 2388 Then train a discriminative classifier (averaged perceptron or logistic regression) on
 2389 the training set, and compute its accuracy and *F*-MEASURE on the test set.
- 2390 Determine whether the differences are statistically significant, using two-tailed hy-
 2391 pothesis tests: Binomial for the difference in accuracy, and bootstrap for the differ-
 2392 ence in macro-*F*-MEASURE.
- 2393 2394 The remaining problems will require you to build a classifier and test its properties. Pick
 2395 a multi-class text classification dataset, such as RCV1¹⁵). Divide your data into training

¹³e.g., https://github.com/google-research-datasets/word_sense_disambiguation_corpora or <http://globalwordnet.org/wordnet-annotated-corpora/>

¹⁴<http://wordnetweb.princeton.edu/perl/webwn>

¹⁵http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm

2396 (60%), development (20%), and test sets (20%), if no such division already exists. [todo:
2397 this dataset is already tokenized, find something else]

2398 7. Compare various vocabulary sizes of $10^2, 10^3, 10^4, 10^5$, using the most frequent words
2399 in each case (you may use any reasonable tokenizer). Train logistic regression clas-
2400 sifiers for each vocabulary size, and apply them to the development set. Plot the
2401 accuracy and Macro-*F*-MEASURE with the increasing vocabulary size. For each vo-
2402 cabulary size, tune the regularizer to maximize accuracy on a subset of data that is
2403 held out from the training set.

2404 8. Compare the following tokenization algorithms:

- 2405 • Whitespace, using a regular expression
2406 • Penn Treebank
2407 • Split input into five-character units, regardless of whitespace or punctuation

2408 Compute the token/type ratio for each tokenizer on the training data, and explain
2409 what you find. Train your classifier on each tokenized dataset, tuning the regularizer
2410 on a subset of data that is held out from the training data. Tokenize the development
2411 set, and report accuracy and Macro-*F*-MEASURE.

2412 9. Apply the Porter and Lancaster stemmers to the training set, using any reasonable
2413 tokenizer, and compute the token/type ratios. Train your classifier on the stemmed
2414 data, and compute the accuracy and Macro-*F*-MEASURE on stemmed development
2415 data, again using a held-out portion of the training data to tune the regularizer.

2416 10. Identify the best combination of vocabulary filtering, tokenization, and stemming
2417 from the previous three problems. Apply this preprocessing to the test set, and
2418 compute the test set accuracy and Macro-*F*-MEASURE. Compare against a baseline
2419 system that applies no vocabulary filtering, whitespace tokenization, and no stem-
2420 ming.

2421 Use the binomial test to determine whether your best-performing system is signifi-
2422 cantly more accurate than the baseline.

2423 Use the bootstrap test with $M = 10^4$ to determine whether your best-performing
2424 system achieves significantly higher macro-*F*-MEASURE.

2425 Chapter 5

2426 Learning without supervision

2427 So far we've assumed the following setup:

- 2428 a **training set** where you get observations x and labels y ;
- 2429 a **test set** where you only get observations x .

2430 Without labeled data, is it possible to learn anything? This scenario is known as **unsu-**
2431 **pervised learning**, and we will see that indeed it is possible to learn about the underlying
2432 structure of unlabeled observations. This chapter will also explore some related scenarios:
2433 **semi-supervised learning**, in which only some instances are labeled, and **domain adap-**
2434 **tation**, in which the training data differs from the data on which the trained system will
2435 be deployed.

2436 5.1 Unsupervised learning

2437 To motivate unsupervised learning, consider the problem of word sense disambiguation
2438 (§ 4.2). Our goal is to classify each instance of a word, such as *bank* into a sense,

- 2439 bank#1: a financial institution
- 2440 bank#2: the land bordering a river

2441 It is difficult to obtain sufficient training data for word sense disambiguation, because
2442 even a large corpus will contain only a few instances of all but the most common words.
2443 Is it possible to learn anything about these different senses without labeled data?

2444 Word sense disambiguation is usually performed using feature vectors constructed
2445 from the local context of the word to be disambiguated. For example, for the word

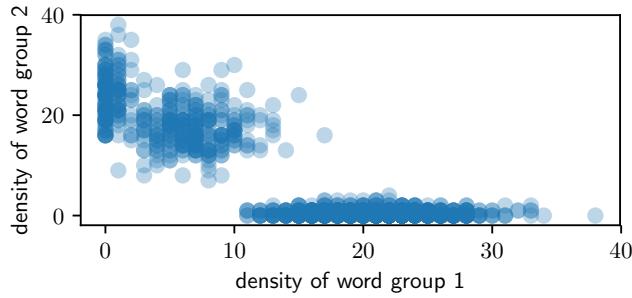


Figure 5.1: Counts of words from two different context groups

2446 *bank*, the immediate context might typically include words from one of the following two
 2447 groups:

- 2448 1. *financial, deposits, credit, lending, capital, markets, regulated, reserve, liquid, assets*
 2449 2. *land, water, geography, stream, river, flow, deposits, discharge, channel, ecology*

2450 Now consider a scatterplot, in which each point is a document containing the word *bank*.
 2451 The location of the document on the x -axis is the count of words in group 1, and the
 2452 location on the y -axis is the count for group 2. In such a plot, shown in Figure 5.1, two
 2453 “blobs” might emerge, and these blobs correspond to the different senses of *bank*.

2454 Here’s a related scenario, from a different problem. Suppose you download thousands
 2455 of news articles, and make a scatterplot, where each point corresponds to a document:
 2456 the x -axis is the frequency of the group of words (*hurricane, winds, storm*); the y -axis is the
 2457 frequency of the group (*election, voters, vote*). This time, three blobs might emerge: one
 2458 for documents that are largely about a hurricane, another for documents largely about a
 2459 election, and a third for documents about neither topic.

2460 These clumps represent the underlying structure of the data. But the two-dimensional
 2461 scatter plots are based on groupings of context words, and in real scenarios these word
 2462 lists are unknown. Unsupervised learning applies the same basic idea, but in a high-
 2463 dimensional space with one dimension for every context word. This space can’t be di-
 2464 rectly visualized, but the idea is the same: try to identify the underlying structure of the
 2465 observed data, such that there are a few clusters of points, each of which is internally
 2466 coherent. **Clustering** algorithms are capable of finding such structure automatically.

2467 5.1.1 **K-means** clustering

2468 Clustering algorithms assign each data point to a discrete cluster, $z_i \in 1, 2, \dots, K$. One of
 2469 the best known clustering algorithms is ***K-means***, an iterative algorithm that maintains

Algorithm 8 K -means clustering algorithm

```

1: procedure  $K$ -MEANS( $\mathbf{x}_{1:N}, K$ )
2:   for  $i \in 1 \dots N$  do                                 $\triangleright$  initialize cluster memberships
3:      $z^{(i)} \leftarrow \text{RandomInt}(1, K)$ 
4:   repeat
5:     for  $k \in 1 \dots K$  do                           $\triangleright$  recompute cluster centers
6:        $\boldsymbol{\nu}_k \leftarrow \frac{1}{\delta(z^{(i)}=k)} \sum_{i=1}^N \delta(z^{(i)} = k) \mathbf{x}^{(i)}$ 
7:     for  $i \in 1 \dots N$  do                       $\triangleright$  reassigned instances to nearest clusters
8:        $z^{(i)} \leftarrow \operatorname{argmin}_k \|\mathbf{x}^{(i)} - \boldsymbol{\nu}_k\|^2$ 
9:   until converged
10:  return  $\{z^{(i)}\}$                                  $\triangleright$  return cluster assignments

```

2470 a cluster assignment for each instance, and a central (“mean”) location for each cluster.
 2471 K -means iterates between updates to the assignments and the centers:

- 2472 1. each instance is placed in the cluster with the closest center;
 2473 2. each center is recomputed as the average over points in the cluster.

2474 This is formalized in Algorithm 8. The term $\|\mathbf{x}^{(i)} - \boldsymbol{\nu}\|^2$ refers to the squared Euclidean
 2475 norm, $\sum_{j=1}^V (x_j^{(i)} - \nu_j)^2$.

2476 **Soft K -means** is a particularly relevant variant. Instead of directly assigning each
 2477 point to a specific cluster, soft K -means assigns each point a **distribution** over clusters
 2478 $\mathbf{q}^{(i)}$, so that $\sum_{k=1}^K q^{(i)}(k) = 1$, and $\forall_k, q^{(i)}(k) \geq 0$. The soft weight $q^{(i)}(k)$ is computed from
 2479 the distance of $\mathbf{x}^{(i)}$ to the cluster center $\boldsymbol{\nu}_k$. In turn, the center of each cluster is computed
 2480 from a **weighted average** of the points in the cluster,

$$\boldsymbol{\nu}_k = \frac{1}{\sum_{i=1}^N q^{(i)}(k)} \sum_{i=1}^N q^{(i)}(k) \mathbf{x}^{(i)}. \quad [5.1]$$

2481 We will now explore a probabilistic version of soft K -means clustering, based on **expectation**
 2482 **maximization** (EM). Because EM clustering can be derived as an approximation to
 2483 maximum-likelihood estimation, it can be extended in a number of useful ways.

2484 5.1.2 Expectation Maximization (EM)

Expectation maximization combines the idea of soft K -means with Naïve Bayes classification. To review, Naïve Bayes defines a probability distribution over the data,

$$\log p(\mathbf{x}, \mathbf{y}; \boldsymbol{\phi}, \boldsymbol{\mu}) = \sum_{i=1}^N \log \left(p(\mathbf{x}^{(i)} | y^{(i)}; \boldsymbol{\phi}) \times p(y^{(i)}; \boldsymbol{\mu}) \right) \quad [5.2]$$

Now suppose that you never observe the labels. To indicate this, we'll refer to the label of each instance as $z^{(i)}$, rather than $y^{(i)}$, which is usually reserved for observed variables. By marginalizing over the **latent** variables \mathbf{z} , we compute the marginal probability of the observed instances \mathbf{x} :

$$\log p(\mathbf{x}; \boldsymbol{\phi}, \boldsymbol{\mu}) = \sum_{i=1}^N \log p(\mathbf{x}^{(i)}; \boldsymbol{\phi}, \boldsymbol{\mu}) \quad [5.3]$$

$$= \sum_{i=1}^N \log \sum_{z=1}^K p(\mathbf{x}^{(i)}, z; \boldsymbol{\phi}, \boldsymbol{\mu}) \quad [5.4]$$

$$= \sum_{i=1}^N \log \sum_{z=1}^K p(\mathbf{x}^{(i)} | z; \boldsymbol{\phi}) \times p(z; \boldsymbol{\mu}). \quad [5.5]$$

2485 To estimate the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$, we can maximize the marginal likelihood in Equation 5.5. Why is this the right thing to maximize? Without labels, discriminative learning 2486 is impossible — there's nothing to discriminate. So maximum likelihood is all we have. 2487

2488 When the labels are observed, we can estimate the parameters of the Naïve Bayes 2489 probability model separately for each label. But marginalizing over the labels couples 2490 these parameters, making direct optimization of $\log p(\mathbf{x})$ intractable. We will approximate 2491 the log-likelihood by introducing an *auxiliary variable* $\mathbf{q}^{(i)}$, which is a distribution over the 2492 label set $\mathcal{Z} = \{1, 2, \dots, K\}$. The optimization procedure will alternate between updates to 2493 \mathbf{q} and updates to the parameters $(\boldsymbol{\phi}, \boldsymbol{\mu})$. Thus, $\mathbf{q}^{(i)}$ plays here as in soft K -means.

To derive the updates for this optimization, multiply the right side of Equation 5.5 by

the ratio $\frac{q^{(i)}(z)}{q^{(i)}(z)} = 1$,

$$\log p(\mathbf{x}; \phi, \mu) = \sum_{i=1}^M \log \sum_{z=1}^K p(\mathbf{x}^{(i)} | z; \phi) \times p(z; \mu) \times \frac{q^{(i)}(z)}{q^{(i)}(z)} \quad [5.6]$$

$$= \sum_{i=1}^M \log \sum_{z=1}^K q^{(i)}(z) \times p(\mathbf{x}^{(i)} | z; \phi) \times p(z; \mu) \times \frac{1}{q^{(i)}(z)} \quad [5.7]$$

$$= \sum_{i=1}^M \log E_{\mathbf{q}^{(i)}} \left[\frac{p(\mathbf{x}^{(i)} | z; \phi) p(z; \mu)}{q^{(i)}(z)} \right], \quad [5.8]$$

where $E_{\mathbf{q}^{(i)}} [f(z)] = \sum_{z=1}^K q^{(i)}(z) \times f(z)$ refers to the expectation of the function f under the distribution $z \sim \mathbf{q}^{(i)}$.

Jensen's inequality says that because \log is a concave function, we can push it inside the expectation, and obtain a lower bound.

$$\log p(\mathbf{x}; \phi, \mu) \geq \sum_{i=1}^N E_{\mathbf{q}^{(i)}} \left[\log \frac{p(\mathbf{x}^{(i)} | z; \phi) p(z; \mu)}{q^{(i)}(z)} \right] \quad [5.9]$$

$$J \triangleq \sum_{i=1}^N E_{\mathbf{q}^{(i)}} \left[\log p(\mathbf{x}^{(i)} | z; \phi) + \log p(z; \mu) - \log q^{(i)}(z) \right] \quad [5.10]$$

$$= \sum_{i=1}^N E_{\mathbf{q}^{(i)}} [\log p(\mathbf{x}^{(i)}, z; \phi, \mu)] + H(\mathbf{q}^{(i)}) \quad [5.11]$$

We will focus on Equation 5.10, which is the lower bound on the marginal log-likelihood of the observed data, $\log p(\mathbf{x})$. Equation 5.11 shows the connection to the information theoretic concept of **entropy**, $H(\mathbf{q}^{(i)}) = -\sum_{z=1}^K q^{(i)}(z) \log q^{(i)}(z)$, which measures the average amount of information produced by a draw from the distribution $q^{(i)}$. The lower bound J is a function of two groups of arguments:

- the distributions $\mathbf{q}^{(i)}$ for each instance;
- the parameters μ and ϕ .

The expectation-maximization (EM) algorithm maximizes the bound with respect to each of these arguments in turn, while holding the other fixed.

5.1.2.1 The E-step

The step in which we update $\mathbf{q}^{(i)}$ is known as the **E-step**, because it updates the distribution under which the expectation is computed. To derive this update, first write out the

expectation in the lower bound as a sum,

$$J = \sum_{i=1}^N \sum_{z=1}^K q^{(i)}(z) \left[\log p(\mathbf{x}^{(i)} | z; \boldsymbol{\phi}) + \log p(z; \boldsymbol{\mu}) - \log q^{(i)}(z) \right]. \quad [5.12]$$

When optimizing this bound, we must also respect a set of “sum-to-one” constraints, $\sum_{z=1}^K q^{(i)}(z) = 1$ for all i . Just as in Naïve Bayes, this constraint can be incorporated into a Lagrangian:

$$J_q = \sum_{i=1}^N \sum_{z=1}^K q^{(i)}(z) \left(\log p(\mathbf{x}^{(i)} | z; \boldsymbol{\phi}) + \log p(z; \boldsymbol{\mu}) - \log q^{(i)}(z) \right) + \lambda^{(i)} \left(1 - \sum_{z=1}^K q^{(i)}(z) \right), \quad [5.13]$$

where $\lambda^{(i)}$ is the Lagrange multiplier for instance i .

The Lagrangian is maximized by taking the derivative and solving for $q^{(i)}$:

$$\frac{\partial J_q}{\partial q^{(i)}(z)} = \log p(\mathbf{x}^{(i)} | z; \boldsymbol{\phi}) + \log p(z; \boldsymbol{\mu}) - \log q^{(i)}(z) - 1 - \lambda^{(i)} \quad [5.14]$$

$$\log q^{(i)}(z) = \log p(\mathbf{x}^{(i)} | z; \boldsymbol{\phi}) + \log p(z; \boldsymbol{\mu}) - 1 - \lambda^{(i)} \quad [5.15]$$

$$q^{(i)}(z) \propto p(\mathbf{x}^{(i)} | z; \boldsymbol{\phi}) \times p(z; \boldsymbol{\mu}). \quad [5.16]$$

Applying the sum-to-one constraint gives an exact solution,

$$q^{(i)}(z) = \frac{p(\mathbf{x}^{(i)} | z; \boldsymbol{\phi}) \times p(z; \boldsymbol{\mu})}{\sum_{z'=1}^K p(\mathbf{x}^{(i)} | z'; \boldsymbol{\phi}) \times p(z'; \boldsymbol{\mu})} \quad [5.17]$$

$$= p(z | \mathbf{x}^{(i)}; \boldsymbol{\phi}, \boldsymbol{\mu}). \quad [5.18]$$

After normalizing, each $q^{(i)}$ — which is the soft distribution over clusters for data $\mathbf{x}^{(i)}$ — is set to the posterior probability $p(z | \mathbf{x}^{(i)}; \boldsymbol{\phi}, \boldsymbol{\mu})$ under the current parameters. Although the Lagrange multipliers $\lambda^{(i)}$ were introduced as additional parameters, they drop out during normalization.

5.1.2.2 The M-step

Next, we hold fixed the soft assignments $q^{(i)}$, and maximize with respect to the parameters, $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$. Let’s focus on the parameter $\boldsymbol{\phi}$, which parametrizes the likelihood $p(\mathbf{x} | z; \boldsymbol{\phi})$, and leave $\boldsymbol{\mu}$ for an exercise. The parameter $\boldsymbol{\phi}$ is a distribution over words for each cluster, so it is optimized under the constraint that $\sum_{j=1}^V \phi_{z,j} = 1$. To incorporate this

constraint, we introduce a set of Lagrange multipliers $\{\lambda_z\}_{z=1}^K$, and from the Lagrangian,

$$J_\phi = \sum_{i=1}^N \sum_{z=1}^K q^{(i)}(z) \left(\log p(\mathbf{x}^{(i)} | z; \boldsymbol{\phi}) + \log p(z; \mu) - \log q^{(i)}(z) \right) + \sum_{z=1}^K \lambda_z \left(1 - \sum_{j=1}^V \phi_{z,j} \right). \quad [5.19]$$

2512 The term $\log p(\mathbf{x}^{(i)} | z; \boldsymbol{\phi})$ is the conditional log-likelihood for the multinomial, which
2513 expands to,

$$\log p(\mathbf{x}^{(i)} | z, \boldsymbol{\phi}) = C + \sum_{j=1}^V x_j \log \phi_{z,j}, \quad [5.20]$$

2514 where C is a constant with respect to $\boldsymbol{\phi}$ — see Equation 2.12 in § 2.1 for more discussion
2515 of this probability function.

Setting the derivative of J_ϕ equal to zero,

$$\frac{\partial J_\phi}{\partial \phi_{z,j}} = \sum_{i=1}^N q^{(i)}(z) \times \frac{x_j^{(i)}}{\phi_{z,j}} - \lambda_z \quad [5.21]$$

$$\phi_{z,j} \propto \sum_{i=1}^N q^{(i)}(z) \times x_j^{(i)}. \quad [5.22]$$

Because ϕ_z is constrained to be a probability distribution, the exact solution is computed as,

$$\phi_{z,j} = \frac{\sum_{i=1}^N q^{(i)}(z) \times x_j^{(i)}}{\sum_{j'=1}^V \sum_{i=1}^N q^{(i)}(z) \times x_{j'}^{(i)}} = \frac{E_q [\text{count}(z, j)]}{\sum_{j'=1}^V E_q [\text{count}(z, j')]} \quad [5.23]$$

2516 where the counter $j \in \{1, 2, \dots, V\}$ indexes over base features, such as words.

2517 This update sets ϕ_z equal to the relative frequency estimate of the *expected counts* under
2518 the distribution q . As in supervised Naïve Bayes, we can smooth these counts by adding
2519 a constant α . The update for μ is similar: $\mu_z \propto \sum_{i=1}^N q^{(i)}(z) = E_q [\text{count}(z)]$, which is the
2520 expected frequency of cluster z . These probabilities can also be smoothed. In sum, the
2521 M-step is just like Naïve Bayes, but with expected counts rather than observed counts.

2522 The multinomial likelihood $p(\mathbf{x} | z)$ can be replaced with other probability distribu-
2523 tions: for example, for continuous observations, a Gaussian distribution can be used. In
2524 some cases, there is no closed-form update to the parameters of the likelihood. One ap-
2525 proach is to run gradient-based optimization at each M-step; another is to simply take a
2526 single step along the gradient step and then return to the E-step (Berg-Kirkpatrick et al.,
2527 2010).

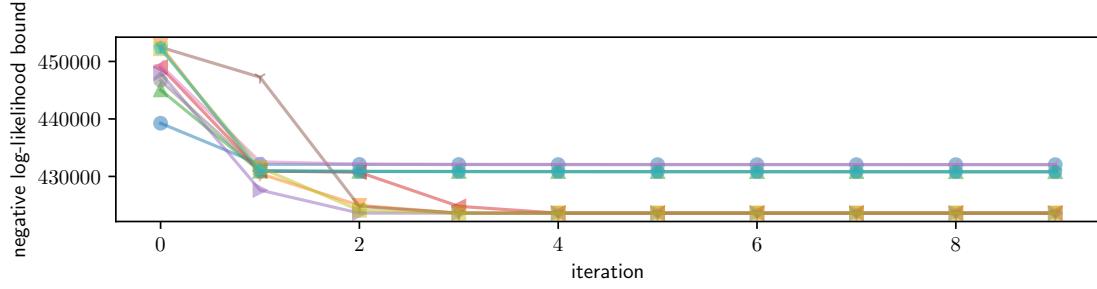


Figure 5.2: Sensitivity of expectation maximization to initialization. Each line shows the progress of optimization from a different random initialization.

2528 5.1.3 EM as an optimization algorithm

2529 Algorithms that alternate between updating subsets of the parameters are called **coordi-**
 2530 **nate ascent** algorithms. The objective J (the lower bound on the marginal likelihood of
 2531 the data) is separately convex in q and (μ, ϕ) , but it is not jointly convex in all terms; this
 2532 condition is known as **biconvexity**. Each step of the expectation-maximization algorithm
 2533 is guaranteed not to decrease the lower bound J , which means that EM will converge
 2534 towards a solution at which no nearby points yield further improvements. This solution
 2535 is a **local optimum** — it is as good or better than any of its immediate neighbors, but is
 2536 *not* guaranteed to be optimal among all possible configurations of (q, μ, ϕ) .

2537 The fact that there is no guarantee of global optimality means that initialization is
 2538 important: where you start can determine where you finish. To illustrate this point,
 2539 Figure 5.2 shows the objective function for EM with ten different random initializations:
 2540 while the objective function improves monotonically in each run, it converges to several
 2541 different values.¹ For the convex objectives that we encountered in chapter 2, it was not
 2542 necessary to worry about initialization, because gradient-based optimization guaranteed
 2543 to reach the global minimum. But in expectation-maximization — and in the deep neural
 2544 networks from chapter 3 — initialization matters.

2545 In **hard EM**, each $q^{(i)}$ distribution assigns probability of 1 to a single label $\hat{z}^{(i)}$, and zero
 2546 probability to all others (Neal and Hinton, 1998). This is similar in spirit to K -means clus-
 2547 tering, and can outperform standard EM in some cases (Spitkovsky et al., 2010). Another
 2548 variant of expectation maximization incorporates stochastic gradient descent (SGD): after
 2549 performing a local E-step at each instance $x^{(i)}$, we immediately make a gradient update
 2550 to the parameters (μ, ϕ) . This algorithm has been called **incremental expectation maxi-**
 2551 **mization** (Neal and Hinton, 1998) and **online expectation maximization** (Sato and Ishii,
 2552 2000; Cappé and Moulines, 2009), and is especially useful when there is no closed-form

¹The figure shows the upper bound on the negative log-likelihood, because optimization is typically framed as minimization rather than maximization.

2553 optimum for the likelihood $p(\mathbf{x} \mid z)$, and in online settings where new data is constantly
 2554 streamed in (see Liang and Klein, 2009, for a comparison for online EM variants).

2555 **5.1.4 How many clusters?**

2556 So far, we have assumed that the number of clusters K is given. In some cases, this as-
 2557 sumption is valid. For example, a lexical semantic resource like WordNet might define the
 2558 number of senses for a word. In other cases, the number of clusters could be a parameter
 2559 for the user to tune: some readers want a coarse-grained clustering of news stories into
 2560 three or four clusters, while others want a fine-grained clustering into twenty or more.
 2561 But many times there is little extrinsic guidance for how to choose K .

2562 One solution is to choose the number of clusters to maximize a metric of clustering
 2563 quality. The other parameters μ and ϕ are chosen to maximize the log-likelihood bound
 2564 J , so this might seem a potential candidate for tuning K . However, J will never decrease
 2565 with K : if it is possible to obtain a bound of J_K with K clusters, then it is always possible
 2566 to do at least as well with $K + 1$ clusters, by simply ignoring the additional cluster and
 2567 setting its probability to zero in q and μ . It is therefore necessary to introduce a penalty
 2568 for model complexity, so that fewer clusters are preferred. For example, the Akaike Infor-
 2569 mation Crition (AIC; Akaike, 1974) is the linear combination of the number of parameters
 2570 and the log-likelihood,

$$\text{AIC} = 2M - 2J, \quad [5.24]$$

2571 where M is the number of parameters. In an expectation-maximization clustering algo-
 2572 rithm, $M = K \times V + K$. Since the number of parameters increases with the number of
 2573 clusters K , the AIC may prefer more parsimonious models, even if they do not fit the data
 2574 quite as well.

2575 Another choice is to maximize the **predictive likelihood** on heldout data. This data
 2576 is not used to estimate the model parameters ϕ and μ , and so it is not the case that the
 2577 likelihood on this data is guaranteed to increase with K . Figure 5.3 shows the negative
 2578 log-likelihood on training and heldout data, as well as the AIC.

2579 ***Bayesian nonparametrics** An alternative approach is to treat the number of clusters as
 2580 another latent variable. This requires statistical inference over a set of models with a vari-
 2581 able number of clusters. This is not possible within the framework of expectation max-
 2582 imization, but there are several alternative inference procedures which can be applied,
 2583 including **Markov Chain Monte Carlo (MCMC)**, which is briefly discussed in § 5.5 (for
 2584 more details, see Chapter 25 of Murphy, 2012). Bayesian nonparametrics have been ap-
 2585 plied to the problem of unsupervised word sense induction, learning not only the word
 2586 senses but also the number of senses per word (Reisinger and Mooney, 2010).

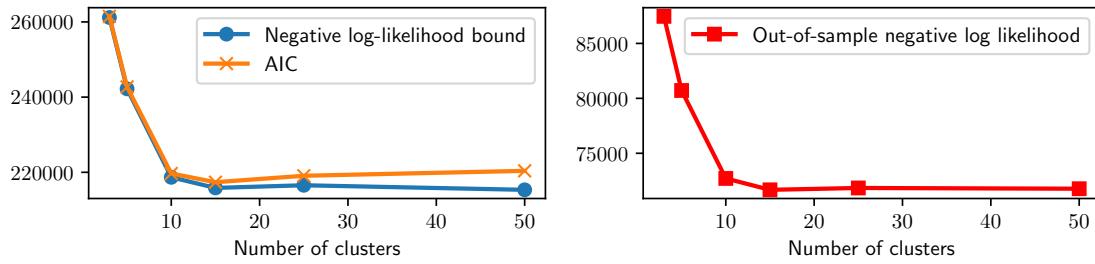


Figure 5.3: The negative log-likelihood and AIC for several runs of expectation maximization, on synthetic data. Although the data was generated from a model with $K = 10$, the optimal number of clusters is $\hat{K} = 15$, according to AIC and the heldout log-likelihood. The training set log-likelihood continues to improve as K increases.

2587 5.2 Applications of expectation-maximization

2588 EM is not really an “algorithm” like, say, quicksort. Rather, it is a framework for learning
2589 with missing data. The recipe for using EM on a problem of interest is:

- 2590 • Introduce latent variables z , such that it is easy to write the probability $P(x, z)$. It
2591 should also be easy to estimate the associated parameters, given knowledge of z .
- 2592 • Derive the E-step updates for $q(z)$, which is typically factored as $q(z) = \prod_{i=1}^N q_{z(i)}(z^{(i)})$,
2593 where i is an index over instances.
- 2594 • The M-step updates typically correspond to the soft version of a probabilistic super-
2595 vised learning algorithm, like Naïve Bayes.

2596 This section discusses a few of the many applications of this general framework.

2597 5.2.1 Word sense induction

2598 The chapter began by considering the problem of word sense disambiguation when the
2599 senses are not known in advance. Expectation-maximization can be applied to this prob-
2600 lem by treating each cluster as a word sense. Each instance represents the use of an
2601 ambiguous word, and $x^{(i)}$ is a vector of counts for the other words that appear nearby:
2602 Schütze (1998) uses all words within a 50-word window. The probability $p(x^{(i)} | z)$ can be
2603 set to the multinomial distribution, as in Naïve Bayes. The EM algorithm can be applied
2604 directly to this data, yielding clusters that (hopefully) correspond to the word senses.

Better performance can be obtained by first applying truncated **singular value decom-
position (SVD)** to the matrix of context-counts $C_{ij} = \text{count}(i, j)$, where $\text{count}(i, j)$ is the

count of word j in the context of instance i . Truncated singular value decomposition approximates the matrix \mathbf{C} as a product of three matrices, $\mathbf{U}, \mathbf{S}, \mathbf{V}$, under the constraint that \mathbf{U} and \mathbf{V} are orthonormal, and \mathbf{S} is diagonal:

$$\begin{aligned} & \min_{\mathbf{U}, \mathbf{S}, \mathbf{V}} \|\mathbf{C} - \mathbf{USV}^\top\|_F \\ & \text{s.t. } \mathbf{U} \in \mathbb{R}^{V \times K}, \mathbf{UU}^\top = \mathbb{I} \\ & \quad \mathbf{S} = \text{Diag}(s_1, s_2, \dots, s_K) \\ & \quad \mathbf{V}^\top \in \mathbb{R}^{N_p \times K}, \mathbf{VV}^\top = \mathbb{I}, \end{aligned} \quad [5.25]$$

where $\|\cdot\|_F$ is the Frobenius norm, $\|X\|_F = \sqrt{\sum_{i,j} X_{i,j}^2}$. The matrix \mathbf{U} contains the left singular vectors of \mathbf{C} , and the rows of this matrix can be used as low-dimensional representations of the count vectors \mathbf{c}_i . EM clustering can be made more robust by setting the instance descriptions $\mathbf{x}^{(i)}$ equal to these rows, rather than using raw counts (Schütze, 1998). However, because the instances are now dense vectors of continuous numbers, the probability $p(\mathbf{x}^{(i)} | z)$ must be defined as a multivariate Gaussian distribution.

In truncated singular value decomposition, the hyperparameter K is the truncation limit: when K is equal to the rank of \mathbf{C} , the norm of the difference between the original matrix \mathbf{C} and its reconstruction \mathbf{USV}^\top will be zero. Lower values of K increase the reconstruction error, but yield vector representations that are smaller and easier to learn from. Singular value decomposition is discussed in more detail in chapter 14.

5.2.2 Semi-supervised learning

Expectation-maximization can also be applied to the problem of **semi-supervised learning**: learning from both labeled and unlabeled data in a single model. Semi-supervised learning makes use of ground truth annotations, ensuring that each label y corresponds to the desired concept. By adding unlabeled data, it is possible cover a greater fraction of the features than would be possible using labeled data alone. Other methods for semi-supervised learning are discussed in § 5.3, but for now, let's approach the problem within the framework of expectation-maximization (Nigam et al., 2000).

Suppose we have labeled data $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N_\ell}$, and unlabeled data $\{\mathbf{x}^{(i)}\}_{i=N_\ell+1}^{N_\ell+N_u}$, where N_ℓ is the number of labeled instances and N_u is the number of unlabeled instances. We can learn from the combined data by maximizing a lower bound on the joint log-likelihood,

$$\mathcal{L} = \sum_{i=1}^{N_\ell} \log p(\mathbf{x}^{(i)}, y^{(i)}; \boldsymbol{\mu}, \boldsymbol{\phi}) + \sum_{j=N_\ell+1}^{N_\ell+N_u} \log p(\mathbf{x}^{(j)}; \boldsymbol{\mu}, \boldsymbol{\phi}) \quad [5.26]$$

$$= \sum_{i=1}^{N_\ell} \left(\log p(\mathbf{x}^{(i)} | y^{(i)}; \boldsymbol{\phi}) + \log p(y^{(i)}; \boldsymbol{\mu}) \right) + \sum_{j=N_\ell+1}^{N_\ell+N_u} \log \sum_{y=1}^K p(\mathbf{x}^{(j)}, y; \boldsymbol{\mu}, \boldsymbol{\phi}). \quad [5.27]$$

Algorithm 9 Generative process for the Naïve Bayes classifier with hidden components

for Document $i \in \{1, 2, \dots, N\}$ **do**:

Draw the label $y^{(i)} \sim \text{Categorical}(\mu)$;

Draw the component $z^{(i)} \sim \text{Categorical}(\beta_{y^{(i)}})$;

Draw the word counts $x^{(i)} | y^{(i)}, z^{(i)} \sim \text{Multinomial}(\phi_{z^{(i)}})$.

2624 The left sum is identical to the objective in Naïve Bayes; the right sum is the marginal log-
 2625 likelihood for expectation-maximization clustering, from Equation 5.5. We can construct a
 2626 lower bound on this log-likelihood by introducing distributions $q^{(j)}$ for all $j \in \{N_\ell + 1, \dots, N_\ell + N_u\}$.
 2627 The E-step updates these distributions; the M-step updates the parameters ϕ and μ , us-
 2628 ing the expected counts from the unlabeled data and the observed counts from the labeled
 2629 data.

2630 A critical issue in semi-supervised learning is how to balance the impact of the labeled
 2631 and unlabeled data on the classifier weights, especially when the unlabeled data is much
 2632 larger than the labeled dataset. The risk is that the unlabeled data will dominate, caus-
 2633 ing the parameters to drift towards a “natural clustering” of the instances — which may
 2634 not correspond to a good classifier for the labeled data. One solution is to heuristically
 2635 reweight the two components of Equation 5.26, tuning the weight of the two components
 2636 on a heldout development set (Nigam et al., 2000).

2637 **5.2.3 Multi-component modeling**

2638 As a final application, let’s return to fully supervised classification. A classic dataset for
 2639 text classification is 20 newsgroups, which contains posts to a set of online forums, called
 2640 newsgroups. One of the newsgroups is `comp.sys.mac.hardware`, which discusses Ap-
 2641 ple computing hardware. Suppose that within this newsgroup there are two kinds of
 2642 posts: reviews of new hardware, and question-answer posts about hardware problems.
 2643 The language in these *components* of the `mac.hardware` class might have little in com-
 2644 mon; if so, it would be better to model these components separately, rather than treating
 2645 their union as a single class. However, the component responsible for each instance is not
 2646 directly observed.

2647 Recall that Naïve Bayes is based on a generative process, which provides a stochastic
 2648 explanation for the observed data. In Naïve Bayes, each label is drawn from a categorical
 2649 distribution with parameter μ , and each vector of word counts is drawn from a multi-
 2650 nomial distribution with parameter ϕ_y . For multi-component modeling, we envision a
 2651 slightly different generative process, incorporating both the observed label $y^{(i)}$ and the
 2652 latent component $z^{(i)}$. This generative process is shown in Algorithm 9. A new parameter
 2653 $\beta_{y^{(i)}}$ defines the distribution of components, conditioned on the label $y^{(i)}$. The component,
 2654 and not the class label, then parametrizes the distribution over words.

-
- (5.1) ☺ Villeneuve a bel et bien **réussi** son pari de changer de perspectives tout en assurant une cohérence à la franchise.²
- (5.2) ☺ Il est également trop **long** et bancal dans sa narration, tiède dans ses intentions, et tirailé entre deux personnages et directions qui ne parviennent pas à coexister en harmonie.³
- (5.3) Denis Villeneuve a **réussi** une suite **parfaitemment** maîtrisée⁴
- (5.4) **Long, bavard**, hyper design, à peine agité (le comble de l'action : une bagarre dans la flotte), métaphysique et, surtout, ennuyeux jusqu'à la catalepsie.⁵
- (5.5) Une suite d'une écrasante puissance, mêlant **parfaitemment** le contemplatif au narratif.⁶
- (5.6) Le film impitoyablement **bavard** finit quand même par se taire quand se lève l'espèce de bouquet final où semble se déchaîner, comme en libre parcours de poulets décapiés, l'armée des graphistes numériques griffant nerveusement la palette graphique entre agonie et orgasme.⁷

Table 5.1: Labeled and unlabeled reviews of the films *Blade Runner 2049* and *Transformers: The Last Knight*.

The labeled data includes $(\mathbf{x}^{(i)}, y^{(i)})$, but not $z^{(i)}$, so this is another case of missing data. Again, we sum over the missing data, applying Jensen's inequality to as to obtain a lower bound on the log-likelihood,

$$\log p(\mathbf{x}^{(i)}, y^{(i)}) = \log \sum_{z=1}^{K_z} p(\mathbf{x}^{(i)}, y^{(i)}, z; \boldsymbol{\mu}, \boldsymbol{\phi}, \boldsymbol{\beta}) \quad [5.28]$$

$$\geq \log p(y^{(i)}; \boldsymbol{\mu}) + E_{q_{Z|Y}^{(i)}} [\log p(\mathbf{x}^{(i)} | z; \boldsymbol{\phi}) + \log p(z | y^{(i)}; \boldsymbol{\beta}) - \log q^{(i)}(z)]. \quad [5.29]$$

We are now ready to apply expectation maximization. As usual, the E-step updates the distribution over the missing data, $q_{Z|Y}^{(i)}$. The M-step updates the parameters,

$$\beta_{y,z} = \frac{E_q [\text{count}(y, z)]}{\sum_{z'=1}^{K_z} E_q [\text{count}(y, z')]} \quad [5.30]$$

$$\phi_{z,j} = \frac{E_q [\text{count}(z, j)]}{\sum_{j'=1}^V E_q [\text{count}(z, j')]} \quad [5.31]$$

2655 5.3 Semi-supervised learning

2656 In semi-supervised learning, the learner makes use of both labeled and unlabeled data.
 2657 To see how this could help, suppose you want to do sentiment analysis in French. In Ta-

ble 5.1, there are two labeled examples, one positive and one negative. From this data, a learner could conclude that *réussi* is positive and *long* is negative. This isn't much! However, we can propagate this information to the unlabeled data, and potentially learn more.

- If we are confident that *réussi* is positive, then we might guess that (5.3) is also positive.
- That suggests that *parfaitement* is also positive.
- We can then propagate this information to (5.5), and learn from this words in this example.
- Similarly, we can propagate from the labeled data to (5.4), which we guess to be negative because it shares the word *long*. This suggests that *bavard* is also negative, which we propagate to (5.6).

Instances (5.3) and (5.4) were “similar” to the labeled examples for positivity and negativity, respectively. By using these instances to expand the models for each class, it became possible to correctly label instances (5.5) and (5.6), which didn't share any important features with the original labeled data. This requires a key assumption: that similar instances will have similar labels.

In § 5.2.2, we discussed how expectation maximization can be applied to semi-supervised learning. Using the labeled data, the initial parameters ϕ would assign a high weight for *réussi* in the positive class, and a high weight for *long* in the negative class. These weights helped to shape the distributions q for instances (5.3) and (5.4) in the E-step. In the next iteration of the M-step, the parameters ϕ are updated with counts from these instances, making it possible to correctly label the instances (5.5) and (5.6).

However, expectation-maximization has an important disadvantage: it requires using a generative classification model, which restricts the features that can be used for classification. In this section, we explore non-probabilistic approaches, which impose fewer restrictions on the classification model.

5.3.1 Multi-view learning

EM semi-supervised learning can be viewed as **self-training**: the labeled data guides the initial estimates of the classification parameters; these parameters are used to compute a label distribution over the unlabeled instances, $q^{(i)}$; the label distributions are used to update the parameters. The risk is that self-training drifts away from the original labeled data. This problem can be ameliorated by **multi-view learning**. Here we take the assumption that the features can be decomposed into multiple “views”, each of which is conditionally independent, given the label. For example, consider the problem of classifying a name as a person or location: one view is the name itself; another is the context in which it appears. This situation is illustrated in Table 5.2.

	$\mathbf{x}^{(1)}$	$\mathbf{x}^{(2)}$	y
1.	Peachtree Street	located on	LOC
2.	Dr. Walker	said	PER
3.	Zanzibar	located in	? → LOC
4.	Zanzibar	flew to	? → LOC
5.	Dr. Robert	recommended	? → PER
6.	Oprah	recommended	? → PER

Table 5.2: Example of multiview learning for named entity classification

2694 **Co-training** is an iterative multi-view learning algorithm, in which there are separate
 2695 classifiers for each view (Blum and Mitchell, 1998). At each iteration of the algorithm, each
 2696 classifier predicts labels for a subset of the unlabeled instances, using only the features
 2697 available in its view. These predictions are then used as ground truth to train the classifiers
 2698 associated with the other views. In the example shown in Table 5.2, the classifier on $\mathbf{x}^{(1)}$
 2699 might correctly label instance #5 as a person, because of the feature *Dr*; this instance would
 2700 then serve as training data for the classifier on $\mathbf{x}^{(2)}$, which would then be able to correctly
 2701 label instance #6, thanks to the feature *recommended*. If the views are truly independent,
 2702 this procedure is robust to drift. Furthermore, it imposes no restrictions on the classifiers
 2703 that can be used for each view.

2704 Word-sense disambiguation is particularly suited to multi-view learning, thanks to the
 2705 heuristic of “one sense per discourse”: if a polysemous word is used more than once in
 2706 a given text or conversation, all usages refer to the same sense (Gale et al., 1992). This
 2707 motivates a multi-view learning approach, in which one view corresponds to the local
 2708 context (the surrounding words), and another view corresponds to the global context at
 2709 the document level (Yarowsky, 1995). The local context view is first trained on a small
 2710 seed dataset. We then identify its most confident predictions on unlabeled instances. The
 2711 global context view is then used to extend these confident predictions to other instances
 2712 within the same documents. These new instances are added to the training data to the
 2713 local context classifier, which is retrained and then applied to the remaining unlabeled
 2714 data.

2715 5.3.2 Graph-based algorithms

2716 Another family of approaches to semi-supervised learning begins by constructing a graph,
 2717 in which pairs of instances are linked with symmetric weights $\omega_{i,j}$, e.g.,

$$\omega_{i,j} = \exp(-\alpha \times \|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2). \quad [5.32]$$

2718 The goal is to use this weighted graph to propagate labels from a small set of labeled
 2719 instances to larger set of unlabeled instances.

2720 In **label propagation**, this is done through a series of matrix operations (Zhu et al.,
 2721 Let \mathbf{Q} be a matrix of size $N \times K$, in which each row $\mathbf{q}^{(i)}$ describes the labeling
 2722 of instance i . When ground truth labels are available, then $\mathbf{q}^{(i)}$ is an indicator vector,
 2723 with $q_{y^{(i)}}^{(i)} = 1$ and $q_{y' \neq y^{(i)}}^{(i)} = 0$. Let us refer to the submatrix of rows containing labeled
 2724 instances as \mathbf{Q}_L , and the remaining rows as \mathbf{Q}_U . The rows of \mathbf{Q}_U are initialized to assign
 2725 equal probabilities to all labels, $q_{i,k} = \frac{1}{K}$.

2726 Now, let $T_{i,j}$ represent the “transition” probability of moving from node j to node i ,

$$T_{i,j} \triangleq \Pr(j \rightarrow i) = \frac{\omega_{i,j}}{\sum_{k=1}^N \omega_{k,j}}. \quad [5.33]$$

We compute values of $T_{i,j}$ for all instances j and all *unlabeled* instances i , forming a matrix of size $N_U \times N$. If the dataset is large, this matrix may be expensive to store and manipulate; a solution is to sparsify it, by keeping only the κ largest values in each row, and setting all other values to zero. We can then “propagate” the label distributions to the unlabeled instances,

$$\tilde{\mathbf{Q}}_U \leftarrow \mathbf{T}\mathbf{Q} \quad [5.34]$$

$$\mathbf{s} \leftarrow \tilde{\mathbf{Q}}_U \mathbf{1} \quad [5.35]$$

$$\mathbf{Q}_U \leftarrow \text{Diag}(\mathbf{s})^{-1} \tilde{\mathbf{Q}}_U. \quad [5.36]$$

2727 The expression $\tilde{\mathbf{Q}}_U \mathbf{1}$ indicates multiplication of $\tilde{\mathbf{Q}}_U$ by a column vector of ones, which is
 2728 equivalent to computing the sum of each row of $\tilde{\mathbf{Q}}_U$. The matrix $\text{Diag}(\mathbf{s})$ is a diagonal
 2729 matrix with the elements of \mathbf{s} on the diagonals. The product $\text{Diag}(\mathbf{s})^{-1} \tilde{\mathbf{Q}}_U$ has the effect
 2730 of normalizing the rows of $\tilde{\mathbf{Q}}_U$, so that each row of \mathbf{Q}_U is a probability distribution over
 2731 labels.

2732 5.4 Domain adaptation

2733 In many practical scenarios, the labeled data differs in some key respect from the data
 2734 to which the trained model is to be applied. A classic example is in consumer reviews:
 2735 we may have labeled reviews of movies (the **source domain**), but we want to predict the
 2736 reviews of appliances (the **target domain**). A similar issues arise with genre differences:
 2737 most linguistically-annotated data is news text, but application domains range from social
 2738 media to electronic health records. In general, there may be several source and target
 2739 domains, each with their own properties; however, for simplicity, this discussion will
 2740 focus mainly on the case of a single source and target domain.

2741 The simplest approach is “direct transfer”: train a classifier on the source domain,
 2742 and apply it directly to the target domain. The accuracy of this approach depends on the
 2743 extent to which features are shared across domains. In review text, words like *outstanding*

and *disappointing* will apply across both movies and appliances; but others, like *terrifying*, may have meanings that are domain-specific. **Domain adaptation** algorithms attempt to do better than direct transfer, by learning from data in both domains. There are two main families of domain adaptation algorithms, depending on whether any labeled data is available in the target domain.

5.4.1 Supervised domain adaptation

In supervised domain adaptation, there is a small amount of labeled data in the target domain, and a large amount of data in the source domain. The simplest approach would be to ignore domain differences, and simply merge the training data from the source and target domains. There are several other baseline approaches to dealing with this scenario (Daumé III, 2007):

Interpolation. Train a classifier for each domain, and combine their predictions. For example,

$$\hat{y} = \operatorname{argmax}_y \lambda_s \Psi_s(\mathbf{x}, y) + (1 - \lambda_s) \Psi_t(\mathbf{x}, y), \quad [5.37]$$

where Ψ_s and Ψ_t are the scoring functions from the source and target domain classifiers respectively, and λ_s is the interpolation weight.

Prediction. Train a classifier on the source domain data, use its prediction as an additional feature in a classifier trained on the target domain data.

Priors. Train a classifier on the source domain data, and use its weights as a prior distribution on the weights of the classifier for the target domain data. This is equivalent to regularizing the target domain weights towards the weights of the source domain classifier (Chelba and Acero, 2006),

$$\ell(\boldsymbol{\theta}_t) = \sum_{i=1}^N \ell^{(i)}(\mathbf{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}_t) + \lambda \|\boldsymbol{\theta}_t - \boldsymbol{\theta}_s\|_2^2, \quad [5.38]$$

where $\ell^{(i)}$ is the prediction loss on instance i , and λ is the regularization weight.

An effective and “frustratingly simple” alternative is EasyAdapt (Daumé III, 2007), which creates copies of each feature: one for each domain and one for the cross-domain setting. For example, a negative review of the film *Wonder Woman* begins, *As boring and flavorless as a three-day-old grilled cheese sandwich....*⁸ The resulting bag-of-words feature

⁸<http://www.colesmithey.com/capsules/2017/06/wonder-woman.HTML>, accessed October 9, 2017.

vector would be,

$$\begin{aligned} \mathbf{f}(\mathbf{x}, y, d) = & \{(boring, -, \text{MOVIE}) : 1, (boring, -, *) : 1, \\ & (flavorless, -, \text{MOVIE}) : 1, (flavorless, -, *) : 1, \\ & (three-day-old, -, \text{MOVIE}) : 1, (three-day-old, -, *) : 1, \\ & \dots\}, \end{aligned}$$

with $(boring, -, \text{MOVIE})$ indicating the word *boring* appearing in a negative labeled document in the MOVIE domain, and $(boring, -, *)$ indicating the same word in a negative labeled document in *any* domain. It is up to the learner to allocate weight between the domain-specific and cross-domain features: for words that facilitate prediction in both domains, the learner will use the cross-domain features; for words that are relevant only to a single domain, the domain-specific features will be used. Any discriminative classifier can be used with these augmented features.⁹

5.4.2 Unsupervised domain adaptation

In unsupervised domain adaptation, there is no labeled data in the target domain. Unsupervised domain adaptation algorithms cope with this problem by trying to make the data from the source and target domains as similar as possible. This is typically done by learning a **projection function**, which puts the source and target data in a shared space, in which a learner can generalize across domains. This projection is learned from data in both domains, and is applied to the base features — for example, the bag-of-words in text classification. The projected features can then be used both for training and for prediction.

5.4.2.1 Linear projection

In linear projection, the cross-domain representation is constructed by a matrix-vector product,

$$\mathbf{g}(\mathbf{x}^{(i)}) = \mathbf{U}\mathbf{x}^{(i)}. \quad [5.39]$$

The projected vectors $\mathbf{g}(\mathbf{x}^{(i)})$ can then be used as base features during both training (from the source domain) and prediction (on the target domain).

The projection matrix \mathbf{U} can be learned in a number of different ways, but many approaches focus on compressing and reconstructing the base features (Ando and Zhang, 2005). For example, we can define a set of **pivot features**, which are typically chosen because they appear in both domains: in the case of review documents, pivot features might include evaluative adjectives like *outstanding* and *disappointing* (Blitzer et al., 2007). For each pivot feature j , we define an auxiliary problem of predicting whether the feature is

⁹EasyAdapt can be explained as a hierarchical Bayesian model, in which the weights for each domain are drawn from a shared prior (Finkel and Manning, 2009).

present in each example, using the remaining base features. Let ϕ_j denote the weights of this classifier, and us horizontally concatenate the weights for each of the N_p pivot features into a matrix $\Phi = [\phi_1, \phi_2, \dots, \phi_{N_p}]$.

We then perform truncated singular value decomposition on Φ , as described in § 5.2.1, obtaining $\Phi \approx \mathbf{U}\mathbf{S}\mathbf{V}^\top$. The rows of the matrix \mathbf{U} summarize information about each base feature: indeed, the truncated singular value decomposition identifies a low-dimension basis for the weight matrix Φ , which in turn links base features to pivot features. Suppose that a base feature *reliable* occurs only in the target domain of appliance reviews. Nonetheless, it will have a positive weight towards some pivot features (e.g., *outstanding*, *recommended*), and a negative weight towards others (e.g., *worthless*, *unpleasant*). A base feature such as *watchable* might have the same associations with the pivot features, and therefore, $\mathbf{u}_{\text{reliable}} \approx \mathbf{u}_{\text{watchable}}$. The matrix \mathbf{U} can thus project the base features into a space in which this information is shared.

5.4.2.2 Non-linear projection

Non-linear transformations of the base features can be accomplished by implementing the transformation function as a deep neural network, which is trained from an auxiliary objective.

Denoising objectives One possibility is to train a projection function to reconstruct a corrupted version of the original input. The original input can be corrupted in various ways: by the addition of random noise (Glorot et al., 2011; Chen et al., 2012), or by the deletion of features (Chen et al., 2012; Yang and Eisenstein, 2015). Denoising objectives share many properties of the linear projection method described above: they enable the projection function to be trained on large amounts of unlabeled data from the target domain, and allow information to be shared across the feature space, thereby reducing sensitivity to rare and domain-specific features.

Adversarial objectives The ultimate goal is for the transformed representations $\mathbf{g}(\mathbf{x}^{(i)})$ to be domain-general. This can be made an explicit optimization criterion by computing the similarity of transformed instances both within and between domains (Tzeng et al., 2015), or by formulating an auxiliary classification task, in which the domain itself is treated as a label (Ganin et al., 2016). This setting is **adversarial**, because we want to learn a representation that makes this classifier perform poorly. At the same time, we want $\mathbf{g}(\mathbf{x}^{(i)})$ to enable accurate predictions of the labels $y^{(i)}$.

To formalize this idea, let $d^{(i)}$ represent the domain of instance i , and let $\ell_d(\mathbf{g}(\mathbf{x}^{(i)}), d^{(i)}; \theta_d)$ represent the loss of a classifier (typically a deep neural network) trained to predict $d^{(i)}$ from the transformed representation $\mathbf{g}(\mathbf{x}^{(i)})$, using parameters θ_d . Analogously, let $\ell_y(\mathbf{g}(\mathbf{x}^{(i)}), y^{(i)}; \theta_y)$ represent the loss of a classifier trained to predict the label $y^{(i)}$ from $\mathbf{g}(\mathbf{x}^{(i)})$, using param-

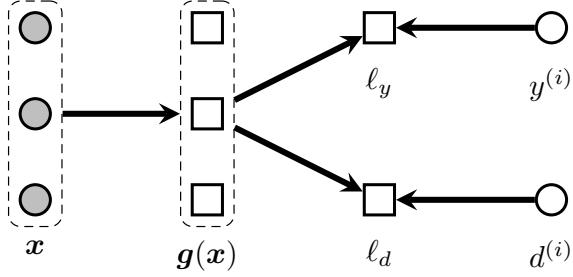


Figure 5.4: A schematic view of adversarial domain adaptation. The loss ℓ_y is computed only for instances from the source domain, where labels $y^{(i)}$ are available.

eters θ_y . The transformation g can then be trained from two criteria: it should yield accurate predictions of the labels $y^{(i)}$, while making *inaccurate* predictions of the domains $d^{(i)}$. This can be formulated as a joint optimization problem,

$$\min_{f, \theta_g, \theta_y, \theta_d} \sum_{i=1}^{N_\ell+N_u} \ell_d(g(\mathbf{x}^{(i)}; \theta_g), d^{(i)}; \theta_d) - \sum_{i=1}^{N_\ell} \ell_y(g(\mathbf{x}^{(i)}), y^{(i)}; \theta_y), \quad [5.40]$$

where N_ℓ is the number of labeled instances and N_u is the number of unlabeled instances, with the labeled instances appearing first in the dataset. This setup is shown in Figure 5.4. The loss can be optimized by stochastic gradient descent, jointly training the parameters of the non-linear transformation θ_g , and the parameters of the prediction models θ_d and θ_y .

5.5 *Other approaches to learning with latent variables

Expectation maximization provides a general approach to learning with latent variables, but it has limitations. One is the sensitivity to initialization; in practical applications, considerable attention may need to be devoted to finding a good initialization. A second issue is that EM tends to be easiest to apply in cases where the latent variables have a clear decomposition (in the cases we have considered, they decompose across the instances). For these reasons, it is worth briefly considering some alternatives to EM.

5.5.1 Sampling

In EM clustering, there is a distribution $q^{(i)}$ for the missing data related to each instance. The M-step consists of updating the parameters of this distribution. An alternative is to draw samples of the latent variables. If the sampling distribution is designed correctly, this procedure will eventually converge to drawing samples from the true posterior over the missing data, $p(z^{(1:N_z)} | \mathbf{x}^{(1:N_x)})$. For example, in the case of clustering, the missing

2849 data $\mathbf{z}^{(1:N_z)}$ is the set of cluster memberships, $\mathbf{y}^{(1:N)}$, so we draw samples from the pos-
 2850 terior distribution over clusterings of the data. If a single clustering is required, we can
 2851 select the one with the highest conditional likelihood, $\hat{\mathbf{z}} = \operatorname{argmax}_{\mathbf{z}} p(\mathbf{z}^{(1:N_z)} | \mathbf{x}^{(1:N_x)})$.

This general family of algorithms is called **Markov Chain Monte Carlo (MCMC)**: “Monte Carlo” because it is based on a series of random draws; “Markov Chain” because the sampling procedure must be designed such that each sample depends only on the previous sample, and not on the entire sampling history. **Gibbs sampling** is an MCMC algorithm in which each latent variable is sampled from its posterior distribution,

$$\mathbf{z}^{(n)} | \mathbf{x}, \mathbf{z}^{(-n)} \sim p(\mathbf{z}^{(n)} | \mathbf{x}, \mathbf{z}^{(-n)}), \quad [5.41]$$

where $\mathbf{z}^{(-n)}$ indicates $\{\mathbf{z} \setminus z^{(n)}\}$, the set of all latent variables except for $z^{(n)}$. Repeatedly drawing samples over all latent variables constructs a Markov chain, and which is guaranteed to converge to a sequence of samples from, $p(\mathbf{z}^{(1:N_z)} | \mathbf{x}^{(1:N_x)})$. In probabilistic clustering, the sampling distribution has the following form,

$$p(z^{(i)} | \mathbf{x}, \mathbf{z}^{(-i)}) = \frac{p(\mathbf{x}^{(i)} | z^{(i)}; \boldsymbol{\phi}) \times p(z^{(i)}; \boldsymbol{\mu})}{\sum_{z=1}^K p(\mathbf{x}^{(i)} | z; \boldsymbol{\phi}) \times p(z; \boldsymbol{\mu})} \quad [5.42]$$

$$\propto \text{Multinomial}(\mathbf{x}^{(i)}; \boldsymbol{\phi}_{z^{(i)}}) \times \boldsymbol{\mu}_{z^{(i)}}. \quad [5.43]$$

2852 In this case, the sampling distribution does not depend on the other instances $\mathbf{x}^{(-i)}, \mathbf{z}^{(-i)}$:
 2853 given the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$, the posterior distribution over each $z^{(i)}$ can be computed
 2854 from $\mathbf{x}^{(i)}$ alone.

2855 In sampling algorithms, there are several choices for how to deal with the parameters.
 2856 One possibility is to sample them too. To do this, we must add them to the generative
 2857 story, by introducing a prior distribution. For the multinomial and categorical parameters
 2858 in the EM clustering model, the **Dirichlet distribution** is a typical choice, since it defines
 2859 a probability on exactly the set of vectors that can be parameters: vectors that sum to one
 2860 and include only non-negative numbers.¹⁰

2861 To incorporate this prior, the generative model must augmented to indicate that each
 2862 $\boldsymbol{\phi}_z \sim \text{Dirichlet}(\boldsymbol{\alpha}_\phi)$, and $\boldsymbol{\mu} \sim \text{Dirichlet}(\boldsymbol{\alpha}_\mu)$. The hyperparameters $\boldsymbol{\alpha}$ are typically set to

¹⁰If $\sum_i^K \theta_i = 1$ and $\theta_i \geq 0$ for all i , then $\boldsymbol{\theta}$ is said to be on the $K - 1$ **simplex**. A Dirichlet distribution with parameter $\boldsymbol{\alpha} \in \mathbb{R}_+^K$ has support over the $K - 1$ simplex,

$$p_{\text{Dirichlet}}(\boldsymbol{\theta} | \boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^K \theta_i^{\alpha_i - 1} \quad [5.44]$$

$$B(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^K \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^K \alpha_i)}, \quad [5.45]$$

with $\Gamma(\cdot)$ indicating the gamma function, a generalization of the factorial function to non-negative reals.

2863 a constant vector $\alpha = [\alpha, \alpha, \dots, \alpha]$. When α is large, the Dirichlet distribution tends to
 2864 generate vectors that are nearly uniform; when α is small, it tends to generate vectors that
 2865 assign most of their probability mass to a few entries. Given prior distributions over ϕ
 2866 and μ , we can now include them in Gibbs sampling, drawing values for these parameters
 2867 from posterior distributions that are conditioned on the other variables in the model.

2868 Unfortunately, sampling ϕ and μ usually leads to slow convergence, meaning that a
 2869 large number of samples is required before the Markov chain breaks free from the initial
 2870 conditions. The reason is that the sampling distributions for these parameters are tightly
 2871 constrained by the cluster memberships $y^{(i)}$, which in turn are tightly constrained by the
 2872 parameters. There are two solutions that are frequently employed:

- 2873 • **Empirical Bayesian** methods maintain ϕ and μ as parameters rather than latent
 2874 variables. They still employ sampling in the E-step of the EM algorithm, but they
 2875 update the parameters using expected counts that are computed from the samples
 2876 rather than from parametric distributions. This EM-MCMC hybrid is also known
 2877 as Monte Carlo Expectation Maximization (MCEM; Wei and Tanner, 1990), and is
 2878 well-suited for cases in which it is difficult to compute $q^{(i)}$ directly.
- 2879 • In **collapsed Gibbs sampling**, we analytically integrate ϕ and μ out of the model.
 2880 The cluster memberships $y^{(i)}$ are the only remaining latent variable; we sample them
 2881 from the compound distribution,

$$p(y^{(i)} | \mathbf{x}^{(1:N)}, \mathbf{y}^{(-i)}; \alpha_\phi, \alpha_\mu) = \int_{\phi, \mu} p(\phi, \mu | \mathbf{y}^{(-i)}, \mathbf{x}^{(1:N)}; \alpha_\phi, \alpha_\mu) p(y^{(i)} | \mathbf{x}^{(1:N)}, \mathbf{y}^{(-i)}, \phi, \mu) d\phi d\mu. \quad [5.46]$$

2882 For multinomial and Dirichlet distributions, the sampling distribution can be com-
 2883 puted in closed form.

2884 MCMC algorithms are guaranteed to converge to the true posterior distribution over
 2885 the latent variables, but there is no way to know how long this will take. In practice, the
 2886 rate of convergence depends on initialization, just as expectation-maximization depends
 2887 on initialization to avoid local optima. Thus, while Gibbs Sampling and other MCMC
 2888 algorithms provide a powerful and flexible array of techniques for statistical inference in
 2889 latent variable models, they are not a panacea for the problems experienced by EM.

2890 5.5.2 Spectral learning

Another approach to learning with latent variables is based on the **method of moments**, which makes it possible to avoid the problem of non-convex log-likelihood. Write $\bar{\mathbf{x}}^{(i)}$ for the normalized vector of word counts in document i , so that $\bar{\mathbf{x}}^{(i)} = \mathbf{x}^{(i)} / \sum_{j=1}^V x_j^{(i)}$. Then

we can form a matrix of word-word co-occurrence probabilities,

$$\mathbf{C} = \sum_{i=1}^N \bar{\mathbf{x}}^{(i)} (\bar{\mathbf{x}}^{(i)})^\top. \quad [5.47]$$

The expected value of this matrix under $p(\mathbf{x} | \phi, \mu)$, as

$$E[\mathbf{C}] = \sum_{i=1}^N \sum_{k=1}^K \Pr(Z^{(i)} = k; \boldsymbol{\mu}) \phi_k \phi_k^\top \quad [5.48]$$

$$= \sum_k^K N \mu_k \phi_k \phi_k^\top \quad [5.49]$$

$$= \Phi \text{Diag}(N\mu) \Phi^\top, \quad [5.50]$$

where Φ is formed by horizontally concatenating $\phi_1 \dots \phi_K$, and $\text{Diag}(N\mu)$ indicates a diagonal matrix with values $N\mu_k$ at position (k, k) . Setting \mathbf{C} equal to its expectation gives,

$$\mathbf{C} = \Phi \text{Diag}(N\mu) \Phi^\top, \quad [5.51]$$

which is similar to the eigendecomposition $\mathbf{C} = \mathbf{Q}\Lambda\mathbf{Q}^\top$. This suggests that simply by finding the eigenvectors and eigenvalues of \mathbf{C} , we could obtain the parameters ϕ and μ , and this is what motivates the name **spectral learning**.

While moment-matching and eigendecomposition are similar in form, they impose different constraints on the solutions: eigendecomposition requires orthonormality, so that $\mathbf{Q}\mathbf{Q}^\top = \mathbb{I}$; in estimating the parameters of a text clustering model, we require that μ and the columns of Φ are probability vectors. Spectral learning algorithms must therefore include a procedure for converting the solution into vectors that are non-negative and sum to one. One approach is to replace eigendecomposition (or the related singular value decomposition) with non-negative matrix factorization (Xu et al., 2003), which guarantees that the solutions are non-negative (Arora et al., 2013).

After obtaining the parameters ϕ and μ , the distribution over clusters can be computed from Bayes' rule:

$$p(z^{(i)} | \mathbf{x}^{(i)}; \phi, \mu) \propto p(\mathbf{x}^{(i)} | z^{(i)}; \phi) \times p(z^{(i)}; \mu). \quad [5.52]$$

Spectral learning yields provably good solutions without regard to initialization, and can be quite fast in practice. However, it is more difficult to apply to a broad family of generative models than more generic techniques like EM and Gibbs Sampling. For more on applying spectral learning across a range of latent variable models, see Anandkumar et al. (2014).

2909 **Additional resources**

2910 There are a number of other learning paradigms that deviate from supervised learning.

- 2911 • **Active learning:** the learner selects unlabeled instances and requests annotations (Set-
- 2912 tles, 2012).
- 2913 • **Multiple instance learning:** labels are applied to bags of instances, with a positive
- 2914 label applied if at least one instance in the bag meets the criterion (Dietterich et al.,
- 2915 1997; Maron and Lozano-Pérez, 1998).
- 2916 • **Constraint-driven learning:** supervision is provided in the form of explicit con-
- 2917 straints on the learner (Chang et al., 2007; Ganchev et al., 2010).
- 2918 • **Distant supervision:** noisy labels are generated from an external resource (Mintz
- 2919 et al., 2009, also see § 17.2.3).
- 2920 • **Multitask learning:** the learner induces a representation that can be used to solve
- 2921 multiple classification tasks (Collobert et al., 2011).
- 2922 • **Transfer learning:** the learner must solve a classification task that differs from the
- 2923 labeled data (Pan and Yang, 2010).

2924 Expectation maximization was introduced by Dempster et al. (1977), and is discussed

2925 in more detail by Murphy (2012). Like most machine learning treatments, Murphy focus

2926 on continuous observations and Gaussian likelihoods, rather than the discrete observa-

2927 tions typically encountered in natural language processing. Murphy (2012) also includes

2928 an excellent chapter on MCMC; for a textbook-length treatment, see Robert and Casella

2929 (2013). For still more on Bayesian latent variable models, see Barber (2012), and for ap-

2930 plications of Bayesian models to natural language processing, see Cohen (2016). Surveys

2931 are available for semi-supervised learning (Zhu and Goldberg, 2009) and domain adapta-

2932 tion (Søgaard, 2013), although both pre-date the current wave of interest in deep learning.

2933 **Exercises**

- 2934 1. Derive the expectation maximization update for the parameter μ in the EM cluster-
- 2935 ing model.
- 2936 2. The expectation maximization lower bound \mathcal{J} is defined in Equation 5.10. Prove
- 2937 that the inverse $-\mathcal{J}$ is convex in q . You can use the following facts about convexity:

 - 2938 • $f(\mathbf{x})$ is convex in \mathbf{x} iff $\alpha f(\mathbf{x}_1) + (1 - \alpha)f(\mathbf{x}_2) \geq f(\alpha\mathbf{x}_1 + (1 - \alpha)\mathbf{x}_2)$ for all
 - 2939 $\alpha \in [0, 1]$.
 - 2940 • If $f(\mathbf{x})$ and $g(\mathbf{x})$ are both convex in \mathbf{x} , then $f(\mathbf{x}) + g(\mathbf{x})$ is also convex in \mathbf{x} .

- 2941 • $\log(x + y) \leq \log x + \log y.$

2942 3. Derive the E-step and M-step updates for the following generative model. You may
2943 assume that the labels $y^{(i)}$ are observed, but $z_m^{(i)}$ is not.

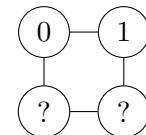
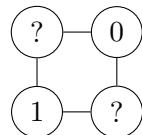
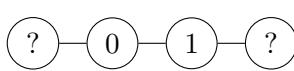
- 2944 • For each instance i ,

- 2945 – Draw label $y^{(i)} \sim \text{Categorical}(\boldsymbol{\mu})$
- 2946 – For each token $m \in \{1, 2, \dots, M^{(i)}\}$
 - 2947 * Draw $z_m^{(i)} \sim \text{Categorical}(\pi)$
 - 2948 * If $z_m^{(i)} = 0$, draw the current token from a label-specific distribution,
2949 $w_m^{(i)} \sim \phi_{y^{(i)}}$
 - 2950 * If $z_m^{(i)} = 1$, draw the current token from a document-specific distribu-
2951 tion, $w_m^{(i)} \sim \nu^{(i)}$

2952 4. Use expectation-maximization clustering to train a word-sense induction system,
2953 applied to the word *say*.

- 2954 • Import `nltk`, run `nltk.download()` and select `semcor`. Import `semcor`
2955 from `nltk.corpus`.
- 2956 • The command `semcor.tagged_sentences(tag='sense')` returns an iter-
2957 ator over sense-tagged sentences in the corpus. Each sentence can be viewed as
2958 an iterator over `tree` objects. For `tree` objects that are sense-annotated words,
2959 you can access the annotation as `tree.label()`, and the word itself with
2960 `tree.leaves()`. So `semcor.tagged_sentences(tag='sense')[0][2].label()`
2961 would return the sense annotation of the third word in the first sentence.
- 2962 • Extract all sentences containing the senses `say.v.01` and `say.v.02`.
- 2963 • Build bag-of-words vectors $x^{(i)}$, containing the counts of other words in those
2964 sentences, including all words that occur in at least two sentences.
- 2965 • Implement and run expectation-maximization clustering on the merged data.
- 2966 • Compute the frequency with which each cluster includes instances of `say.v.01`
2967 and `say.v.02`.

2968 5. Using the iterative updates in Equations 5.34-5.36, compute the outcome of the label
2969 propagation algorithm for the following examples.



2970 The value inside the node indicates the label, $y^{(i)} \in \{0, 1\}$, with $y^{(i)} = ?$ for unlabeled
 2971 nodes. The presence of an edge between two nodes indicates $w_{i,j} = 1$, and the
 2972 absence of an edge indicates $w_{i,j} = 0$. For the third example, you need only compute
 2973 the first three iterations, and then you can guess at the solution in the limit.

2974 In the remaining exercises, you will try out some approaches for semisupervised learning
 2975 and domain adaptation. You will need datasets in multiple domains. You can obtain
 2976 product reviews in multiple domains here: https://www.cs.jhu.edu/~mdredze/datasets/sentiment/processed_acl.tar.gz. Choose a source and target domain,
 2977 e.g. dvds and books, and divide the data for the target domain into training and test sets
 2978 of equal size.

- 2980 6. First, quantify the cost of cross-domain transfer.
- 2981 • Train a logistic regression classifier on the source domain training set, and eval-
 uate it on the target domain test set.
 - 2983 • Train a logistic regression classifier on the target domain training set, and eval-
 uate it on the target domain test set. This is the “direct transfer” baseline.

2985 Compute the difference in accuracy, which is a measure of the transfer loss across
 2986 domains.

- 2987 7. Next, apply the **label propagation** algorithm from § 5.3.2.

2988 As a baseline, using only 5% of the target domain training set, train a classifier, and
 2989 compute its accuracy on the target domain test set.

2990 Next, apply label propagation:

- 2991 • Compute the label matrix \mathbf{Q}_L for the labeled data (5% of the target domain
 training set), with each row equal to an indicator vector for the label (positive
 or negative).
- 2994 • Iterate through the target domain instances, including both test and training
 data. At each instance i , compute all w_{ij} , using Equation 5.32, with $\alpha = 0.01$.
 Use these values to fill in column i of the transition matrix \mathbf{T} , setting all but the
 ten largest values to zero for each column i . Be sure to normalize the column
 so that the remaining values sum to one. You may need to use a sparse matrix
 for this to fit into memory.
- 3000 • Apply the iterative updates from Equations 5.34-5.36 to compute the outcome
 of the label propagation algorithm for the unlabeled examples.

3002 Select the test set instances from \mathbf{Q}_U , and compute the accuracy of this method.
 3003 Compare with the supervised classifier trained only on the 5% sample of the target
 3004 domain training set.

- 3005 8. Using only 5% of the target domain training data (and all of the source domain train-
3006 ing data), implement one of the supervised domain adaptation baselines in § 5.4.1.
3007 See if this improves on the “direct transfer” baseline from the previous problem
- 3008 9. Implement EasyAdapt (§ 5.4.1), again using 5% of the target domain training data
3009 and all of the source domain data.
- 3010 10. Now try unsupervised domain adaptation, using the “linear projection” method
3011 described in § 5.4.2. Specifically:
- 3012 • Identify 500 pivot features as the words with the highest frequency in the (com-
3013 plete) training data for the source and target domains. Specifically, let x_i^d be the
3014 count of the word i in domain d : choose the 500 words with the largest values
3015 of $\min(x_i^{\text{source}}, x_i^{\text{target}})$.
 - 3016 • Train a classifier to predict each pivot feature from the remaining words in the
3017 document.
 - 3018 • Arrange the features of these classifiers into a matrix Φ , and perform truncated
3019 singular value decomposition, with $k = 20$
 - 3020 • Train a classifier from the source domain data, using the combined features
3021 $\mathbf{x}^{(i)} \oplus \mathbf{U}^\top \mathbf{x}^{(i)}$ — these include the original bag-of-words features, plus the pro-
3022 jected features.
 - 3023 • Apply this classifier to the target domain test set, and compute the accuracy.

3024

Part II

3025

Sequences and trees

3026

Chapter 6

3027

Language models

3028 In probabilistic classification, the problem is to compute the probability of a label, conditioned
3029 on the text. Let's now consider the inverse problem: computing the probability of
3030 text itself. Specifically, we will consider models that assign probability to a sequence of
3031 word tokens, $p(w_1, w_2, \dots, w_M)$, with $w_m \in \mathcal{V}$. The set \mathcal{V} is a discrete vocabulary,

$$\mathcal{V} = \{aardvark, abacus, \dots, zither\}. \quad [6.1]$$

3032 Why would you want to compute the probability of a word sequence? In many applications,
3033 the goal is to produce word sequences as output:

- 3034 • In **machine translation** (chapter 18), we convert from text in a source language to
3035 text in a target language.
- 3036 • In **speech recognition**, we convert from audio signal to text.
- 3037 • In **summarization** (§ 16.3.4.1; § 19.2), we convert from long texts into short texts.
- 3038 • In **dialogue systems** (§ 19.3), we convert from the user's input (and perhaps an
3039 external knowledge base) into a text response.

3040 In many of the systems for performing these tasks, there is a subcomponent that computes
3041 the probability of the output text. The purpose of this component is to generate
3042 texts that are more **fluent**. For example, suppose we want to translate a sentence from
3043 Spanish to English.

3044 (6.1) El cafe negro me gusta mucho.

3045 Here is a literal word-for-word translation (a **gloss**):

3046 (6.2) The coffee black me pleases much.

3047 A good language model of English will tell us that the probability of this translation is
 3048 low, in comparison with more grammatical alternatives,

$$p(\text{The coffee black me pleases much}) < p(\text{I love dark coffee}). \quad [6.2]$$

3049 How can we use this fact? Warren Weaver, one of the early leaders in machine trans-
 3050 lation, viewed it as a problem of breaking a secret code (Weaver, 1955):

3051 When I look at an article in Russian, I say: 'This is really written in English,
 3052 but it has been coded in some strange symbols. I will now proceed to decode.'

3053 This observation motivates a generative model (like Naïve Bayes):

3054 • The English sentence $w^{(e)}$ is generated from a **language model**, $p_e(w^{(e)})$.

3055 • The Spanish sentence $w^{(s)}$ is then generated from a **translation model**, $p_{s|e}(w^{(s)} | w^{(e)})$.

Given these two distributions, we can then perform translation by Bayes rule:

$$p_{e|s}(w^{(e)} | w^{(s)}) \propto p_{e,s}(w^{(e)}, w^{(s)}) \quad [6.3]$$

$$= p_{s|e}(w^{(s)} | w^{(e)}) \times p_e(w^{(e)}). \quad [6.4]$$

3056 This is sometimes called the **noisy channel model**, because it envisions English text
 3057 turning into Spanish by passing through a noisy channel, $p_{s|e}$. What is the advantage of
 3058 modeling translation this way, as opposed to modeling $p_{e|s}$ directly? The crucial point is
 3059 that the two distributions $p_{s|e}$ (the translation model) and p_e (the language model) can be
 3060 estimated from separate data. The translation model requires examples of correct trans-
 3061 lations, but the language model requires only text in English. Such monolingual data is
 3062 much more widely available. Furthermore, once estimated, the language model p_e can be
 3063 reused in any application that involves generating English text, from summarization to
 3064 speech recognition.

3065 6.1 *N*-gram language models

A simple approach to computing the probability of a sequence of tokens is to use a **relative frequency estimate**. For example, consider the quote, attributed to Picasso, "computers are useless, they can only give you answers." We can estimate the probability of this sentence,

$$\begin{aligned} p(\text{Computers are useless, they can only give you answers}) \\ = \frac{\text{count}(\text{Computers are useless, they can only give you answers})}{\text{count(all sentences ever spoken)}} \end{aligned} \quad [6.5]$$

3066 This estimator is **unbiased**: in the theoretical limit of infinite data, the estimate will
 3067 be correct. But in practice, we are asking for accurate counts over an infinite number of
 3068 events, since sequences of words can be arbitrarily long. Even with an aggressive upper
 3069 bound of, say, $M = 20$ tokens in the sequence, the number of possible sequences is V^{20} . A
 3070 small vocabulary for English would have $V = 10^4$, so there are 10^{80} possible sequences.
 3071 Clearly, this estimator is very data-hungry, and suffers from high variance: even gram-
 3072 matical sentences will have probability zero if have not occurred in the training data.¹ We
 3073 therefore need to introduce bias to have a chance of making reliable estimates from finite
 3074 training data. The language models that follow in this chapter introduce bias in various
 3075 ways.

We begin with n -gram language models, which compute the probability of a sequence as the product of probabilities of subsequences. The probability of a sequence $p(w) = p(w_1, w_2, \dots, w_M)$ can be refactored using the chain rule (see § A.2):

$$p(w) = p(w_1, w_2, \dots, w_M) \quad [6.6]$$

$$= p(w_1) \times p(w_2 | w_1) \times p(w_3 | w_2, w_1) \times \dots \times p(w_M | w_{M-1}, \dots, w_1) \quad [6.7]$$

Each element in the product is the probability of a word given all its predecessors. We can think of this as a *word prediction* task: given the context *Computers are*, we want to compute a probability over the next token. The relative frequency estimate of the probability of the word *useless* in this context is,

$$\begin{aligned} p(\text{useless} | \text{computers are}) &= \frac{\text{count}(\text{computers are useless})}{\sum_{x \in \mathcal{V}} \text{count}(\text{computers are } x)} \\ &= \frac{\text{count}(\text{computers are useless})}{\text{count}(\text{computers are})}. \end{aligned}$$

3076 We haven't made any approximations yet, and we could have just as well applied the
 3077 chain rule in reverse order,

$$p(w) = p(w_M) \times p(w_{M-1} | w_M) \times \dots \times p(w_1 | w_2, \dots, w_M), \quad [6.8]$$

3078 or in any other order. But this means that we also haven't really made any progress:
 3079 to compute the conditional probability $p(w_M | w_{M-1}, w_{M-2}, \dots, w_1)$, we would need to
 3080 model V^{M-1} contexts. Such a distribution cannot be estimated from any realistic sample
 3081 of text.

¹Chomsky has famously argued that this is evidence against the very concept of probabilistic language models: no such model could distinguish the grammatical sentence *colorless green ideas sleep furiously* from the ungrammatical permutation *furiously sleep ideas green colorless*. Indeed, even the bigrams in these two examples are unlikely to occur — at least, not in texts written before Chomsky proposed this example.

To solve this problem, n -gram models make a crucial simplifying approximation: condition on only the past $n - 1$ words.

$$p(w_m | w_{m-1} \dots w_1) \approx p(w_m | w_{m-1}, \dots, w_{m-n+1}) \quad [6.9]$$

This means that the probability of a sentence w can be approximated as

$$p(w_1, \dots, w_M) \approx \prod_m^M p(w_m | w_{m-1}, \dots, w_{m-n+1}) \quad [6.10]$$

To compute the probability of an entire sentence, it is convenient to pad the beginning and end with special symbols \square and \blacksquare . Then the bigram ($n = 2$) approximation to the probability of *I like black coffee* is:

$$p(I \text{ like black coffee}) = p(I | \square) \times p(\text{like} | I) \times p(\text{black} | \text{like}) \times p(\text{coffee} | \text{black}) \times p(\blacksquare | \text{coffee}). \quad [6.11]$$

3082 This model requires estimating and storing the probability of only V^n events, which is
 3083 exponential in the order of the n -gram, and not V^M , which is exponential in the length of
 3084 the sentence. The n -gram probabilities can be computed by relative frequency estimation,

$$p(w_m | w_{m-1}, w_{m-2}) = \frac{\text{count}(w_{m-2}, w_{m-1}, w_m)}{\sum_{w'} \text{count}(w_{m-2}, w_{m-1}, w')} \quad [6.12]$$

3085 The hyperparameter n controls the size of the context used in each conditional proba-
 3086 bility. If this is misspecified, the language model will perform poorly. Let's consider the
 3087 potential problems concretely.

3088 **When n is too small.** Consider the following sentences:

- 3089 (6.3) **Gorillas** always like to groom **their** friends.
 3090 (6.4) The **computer** that's on the 3rd floor of our office building **crashed**.

3091 In each example, the bolded words depend on each other: the likelihood of *their*
 3092 depends on knowing that *gorillas* is plural, and the likelihood of *crashed* depends on
 3093 knowing that the subject is a *computer*. If the n -grams are not big enough to capture
 3094 this context, then the resulting language model would offer probabilities that are too
 3095 low for these sentences, and too high for sentences that fail basic linguistic tests like
 3096 number agreement.

3097 **When n is too big.** In this case, it is hard to get good estimates of the n -gram parameters from
 3098 our dataset, because of data sparsity. To handle the *gorilla* example, it is necessary to
 3099 model 6-grams, which means accounting for V^6 events. Under a very small vocab-
 3100 uary of $V = 10^4$, this means estimating the probability of 10^{24} distinct events.

3101 These two problems point to another **bias-variance tradeoff** (see § 2.1.4). A small n -
 3102 gram size introduces high bias, and a large n -gram size introduces high variance. But
 3103 in reality we often have both problems at the same time! Language is full of long-range
 3104 dependencies that we cannot capture because n is too small; at the same time, language
 3105 datasets are full of rare phenomena, whose probabilities we fail to estimate accurately
 3106 because n is too large. One solution is to try to keep n large, while still making low-
 3107 variance estimates of the underlying parameters. To do this, we will introduce a different
 3108 sort of bias: **smoothing**.

3109 **6.2 Smoothing and discounting**

3110 Limited data is a persistent problem in estimating language models. In § 6.1, we presented
 3111 n -grams as a partial solution. sparse data can be a problem even for low-order n -grams;
 3112 at the same time, many linguistic phenomena, like subject-verb agreement, cannot be in-
 3113 corporated into language models without high-order n -grams. It is therefore necessary to
 3114 add additional inductive biases to n -gram language models. This section covers some of
 3115 the most intuitive and common approaches, but there are many more (Chen and Good-
 3116 man, 1999).

3117 **6.2.1 Smoothing**

3118 A major concern in language modeling is to avoid the situation $p(w) = 0$, which could
 3119 arise as a result of a single unseen n-gram. A similar problem arose in Naïve Bayes, and
 3120 the solution was **smoothing**: adding imaginary “pseudo” counts. The same idea can be
 3121 applied to n -gram language models, as shown here in the bigram case,

$$P_{\text{smooth}}(w_m \mid w_{m-1}) = \frac{\text{count}(w_{m-1}, w_m) + \alpha}{\sum_{w' \in \mathcal{V}} \text{count}(w_{m-1}, w') + V\alpha}. \quad [6.13]$$

3122 This basic framework is called **Lidstone smoothing**, but special cases have other names:

- 3123 • **Laplace smoothing** corresponds to the case $\alpha = 1$.
- 3124 • **Jeffreys-Perks law** corresponds to the case $\alpha = 0.5$. Manning and Schütze (1999)
 3125 offer more insight on the justifications for this setting.

3126 To maintain normalization, anything that we add to the numerator (α) must also ap-
 3127 pear in the denominator ($V\alpha$). This idea is reflected in the concept of **effective counts**:

$$c_i^* = (c_i + \alpha) \frac{M}{M + V\alpha}, \quad [6.14]$$

	counts	unsmoothed probability	Lidstone smoothing, $\alpha = 0.1$		Discounting, $d = 0.1$	
			effective counts	smoothed probability	effective counts	smoothed probability
<i>impropriety</i>	8	0.4	7.826	0.391	7.9	0.395
<i>offense</i>	5	0.25	4.928	0.246	4.9	0.245
<i>damage</i>	4	0.2	3.961	0.198	3.9	0.195
<i>deficiencies</i>	2	0.1	2.029	0.101	1.9	0.095
<i>outbreak</i>	1	0.05	1.063	0.053	0.9	0.045
<i>infirmity</i>	0	0	0.097	0.005	0.25	0.013
<i>cephalopods</i>	0	0	0.097	0.005	0.25	0.013

Table 6.1: Example of Lidstone smoothing and absolute discounting in a bigram language model, for the context (*alleged*, *_*), for a toy corpus with a total of twenty counts over the seven words shown. Note that discounting decreases the probability for all but the unseen words, while Lidstone smoothing increases the effective counts and probabilities for *deficiencies* and *outbreak*.

where c_i is the count of event i , c_i^* is the effective count, and $M = \sum_{i=1}^V c_i$ is the total number of tokens in the dataset (w_1, w_2, \dots, w_M) . This term ensures that $\sum_{i=1}^V c_i^* = \sum_{i=1}^V c_i = M$. The **discount** for each n-gram is then computed as,

$$d_i = \frac{c_i^*}{c_i} = \frac{(c_i + \alpha)}{c_i} \frac{M}{(M + V\alpha)}.$$

3128 6.2.2 Discounting and backoff

3129 Discounting “borrows” probability mass from observed n -grams and redistributes it. In
 3130 Lidstone smoothing, the borrowing is done by increasing the denominator of the relative
 3131 frequency estimates. The borrowed probability mass is then redistributed by increasing
 3132 the numerator for all n -grams. Another approach would be to borrow the same amount
 3133 of probability mass from all observed n -grams, and redistribute it among only the unob-
 3134 served n -grams. This is called **absolute discounting**. For example, suppose we set an
 3135 absolute discount $d = 0.1$ in a bigram model, and then redistribute this probability mass
 3136 equally over the unseen words. The resulting probabilities are shown in Table 6.1.

Discounting reserves some probability mass from the observed data, and we need not redistribute this probability mass equally. Instead, we can **backoff** to a lower-order language model: if you have trigrams, use trigrams; if you don’t have trigrams, use bigrams; if you don’t even have bigrams, use unigrams. This is called **Katz backoff**. In the simple

case of backing off from bigrams to unigrams, the bigram probabilities are computed as,

$$c^*(i, j) = c(i, j) - d \quad [6.15]$$

$$p_{\text{Katz}}(i | j) = \begin{cases} \frac{c^*(i, j)}{c(j)} & \text{if } c(i, j) > 0 \\ \alpha(j) \times \frac{p_{\text{unigram}}(i)}{\sum_{i': c(i', j)=0} p_{\text{unigram}}(i')} & \text{if } c(i, j) = 0. \end{cases} \quad [6.16]$$

3137 The term $\alpha(j)$ indicates the amount of probability mass that has been discounted for
 3138 context j . This probability mass is then divided across all the unseen events, $\{i' : c(i', j) =$
 3139 $0\}$, proportional to the unigram probability of each word i' . The discount parameter d can
 3140 be optimized to maximize performance (typically held-out log-likelihood) on a develop-
 3141 ment set.

3142 6.2.3 *Interpolation

3143 Backoff is one way to combine different order n -gram models. An alternative approach
 3144 is **interpolation**: setting the probability of a word in context to a weighted sum of its
 3145 probabilities across progressively shorter contexts.

Instead of choosing a single n for the size of the n -gram, we can take the weighted average across several n -gram probabilities. For example, for an interpolated trigram model,

$$\begin{aligned} p_{\text{Interpolation}}(w_m | w_{m-1}, w_{m-2}) &= \lambda_3 p_3^*(w_m | w_{m-1}, w_{m-2}) \\ &\quad + \lambda_2 p_2^*(w_m | w_{m-1}) \\ &\quad + \lambda_1 p_1^*(w_m). \end{aligned}$$

3146 In this equation, p_n^* is the unsmoothed empirical probability given by an n -gram lan-
 3147 guage model, and λ_n is the weight assigned to this model. To ensure that the interpolated
 3148 $p(w)$ is still a valid probability distribution, the values of λ must obey the constraint,
 3149 $\sum_{n=1}^{n_{\max}} \lambda_n = 1$. But how to find the specific values?

3150 An elegant solution is **expectation maximization**. Recall from chapter 5 that we can
 3151 think about EM as learning with *missing data*: we just need to choose missing data such
 3152 that learning would be easy if it weren't missing. What's missing in this case? Think of
 3153 each word w_m as drawn from an n -gram of unknown size, $z_m \in \{1 \dots n_{\max}\}$. This z_m is
 3154 the missing data that we are looking for. Therefore, the application of EM to this problem
 3155 involves the following **generative process**:

3156 **for** Each token $w_m, m = 1, 2, \dots, M$ **do**:
 3157 draw the n -gram size $z_m \sim \text{Categorical}(\lambda)$;
 3158 draw $w_m \sim p_{z_m}^*(w_m | w_{m-1}, \dots, w_{m-z_m})$.

If the missing data $\{Z_m\}$ were known, then λ could be estimated as the relative frequency,

$$\lambda_z = \frac{\text{count}(Z_m = z)}{M} \quad [6.17]$$

$$\propto \sum_{m=1}^M \delta(Z_m = z). \quad [6.18]$$

But since we do not know the values of the latent variables Z_m , we impute a distribution q_m in the E-step, which represents the degree of belief that word token w_m was generated from a n -gram of order z_m ,

$$q_m(z) \triangleq \Pr(Z_m = z \mid \mathbf{w}_{1:m}; \lambda) \quad [6.19]$$

$$= \frac{p(w_m \mid \mathbf{w}_{1:m-1}, Z_m = z) \times p(z)}{\sum_{z'} p(w_m \mid \mathbf{w}_{1:m-1}, Z_m = z') \times p(z')} \quad [6.20]$$

$$\propto p_z^*(w_m \mid \mathbf{w}_{1:m-1}) \times \lambda_z. \quad [6.21]$$

In the M-step, λ is computed by summing the expected counts under q ,

$$\lambda_z \propto \sum_{m=1}^M q_m(z). \quad [6.22]$$

3160 A solution is obtained by iterating between updates to q and λ . The complete algorithm
 3161 is shown in Algorithm 10.

Algorithm 10 Expectation-maximization for interpolated language modeling

```

1: procedure ESTIMATE INTERPOLATED  $n$ -GRAM ( $\mathbf{w}_{1:M}, \{p_n^*\}_{n \in 1:n_{\max}}$ )
2:   for  $z \in \{1, 2, \dots, n_{\max}\}$  do ▷ Initialization
3:      $\lambda_z \leftarrow \frac{1}{n_{\max}}$ 
4:   repeat
5:     for  $m \in \{1, 2, \dots, M\}$  do ▷ E-step
6:       for  $z \in \{1, 2, \dots, n_{\max}\}$  do
7:          $q_m(z) \leftarrow p_z^*(w_m \mid \mathbf{w}_{1:m-1}) \times \lambda_z$ 
8:        $q_m \leftarrow \text{Normalize}(q_m)$ 
9:     for  $z \in \{1, 2, \dots, n_{\max}\}$  do ▷ M-step
10:       $\lambda_z \leftarrow \frac{1}{M} \sum_{m=1}^M q_m(z)$ 
11:    until tired
12:    return  $\lambda$ 
  
```

3162 **6.2.4 *Kneser-Ney smoothing**

3163 Kneser-Ney smoothing is based on absolute discounting, but it redistributes the result-
 3164 ing probability mass in a different way from Katz backoff. Empirical evidence points
 3165 to Kneser-Ney smoothing as the state-of-art for n -gram language modeling (Goodman,
 3166 2001). To motivate Kneser-Ney smoothing, consider the example: *I recently visited ..*
 3167 Which of the following is more likely?

- 3168 • *Francisco*
 3169 • *Duluth*

3170 Now suppose that both bigrams *visited Duluth* and *visited Francisco* are unobserved in
 3171 the training data, and furthermore, the unigram probability $p_1^*(\text{Francisco})$ is greater than
 3172 $p^*(\text{Duluth})$. Nonetheless we would still guess that $p(\text{visited Duluth}) > p(\text{visited Francisco})$,
 3173 because *Duluth* is a more “versatile” word: it can occur in many contexts, while *Francisco*
 3174 usually occurs in a single context, following the word *San*. This notion of versatility is the
 3175 key to Kneser-Ney smoothing.

Writing u for a context of undefined length, and $\text{count}(w, u)$ as the count of word w in
 context u , we define the Kneser-Ney bigram probability as

$$p_{KN}(w | u) = \begin{cases} \frac{\text{count}(w, u) - d}{\text{count}(u)}, & \text{count}(w, u) > 0 \\ \alpha(u) \times p_{\text{continuation}}(w), & \text{otherwise} \end{cases} \quad [6.23]$$

$$p_{\text{continuation}}(w) = \frac{|u : \text{count}(w, u) > 0|}{\sum_{u' \in \mathcal{V}} |u' : \text{count}(w', u') > 0|}. \quad [6.24]$$

First, note that we reserve probability mass using absolute discounting d , which is taken from all unobserved n -grams. The total amount of discounting in context u is $d \times |w : \text{count}(w, u) > 0|$, and we divide this probability mass equally among the unseen n -grams,

$$\alpha(u) = |w : \text{count}(w, u) > 0| \times \frac{d}{\text{count}(u)}. \quad [6.25]$$

3176 This is the amount of probability mass left to account for versatility, which we define via
 3177 the *continuation probability* $p_{\text{continuation}}(w)$ as proportional to the number of observed con-
 3178 texts in which w appears. The numerator of the continuation probability is the number of
 3179 contexts u in which w appears; the denominator normalizes the probability by summing
 3180 the same quantity over all words w' .

3181 The idea of modeling versatility by counting contexts may seem heuristic, but there is
 3182 an elegant theoretical justification from Bayesian nonparametrics (Teh, 2006). Kneser-Ney
 3183 smoothing on n -grams was the dominant language modeling technique before the arrival
 3184 of neural language models.

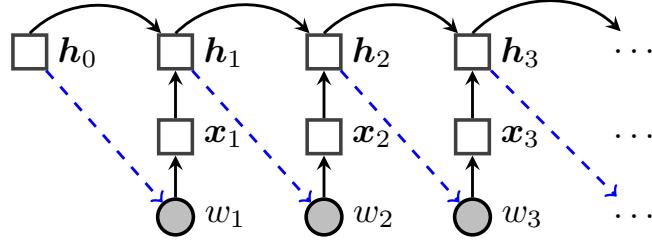


Figure 6.1: The recurrent neural network language model, viewed as an “unrolled” computation graph. Solid lines indicate direct computation, dotted blue lines indicate probabilistic dependencies, circles indicate random variables, and squares indicate computation nodes.

3185 6.3 Recurrent neural network language models

3186 N -gram language models have been largely supplanted by **neural networks**. These mod-
 3187 els do not make the n -gram assumption of restricted context; indeed, they can incorporate
 3188 arbitrarily distant contextual information, while remaining computationally and statisti-
 3189 cally tractable.

3190 The first insight behind neural language models is to treat word prediction as a *dis-
 3191 criminative learning task*.² The goal is to compute the probability $p(w | u)$, where $w \in \mathcal{V}$ is
 3192 a word, and u is the context, which depends on the previous words. Rather than directly
 3193 estimating the word probabilities from (smoothed) relative frequencies, we can treat
 3194 language modeling as a machine learning problem, and estimate parameters that maxi-
 3195 mize the log conditional probability of a corpus.

3196 The second insight is to reparametrize the probability distribution $p(w | u)$ as a func-
 3197 tion of two dense K -dimensional numerical vectors, $\beta_w \in \mathbb{R}^K$, and $v_u \in \mathbb{R}^K$,

$$p(w | u) = \frac{\exp(\beta_w \cdot v_u)}{\sum_{w' \in \mathcal{V}} \exp(\beta_{w'} \cdot v_u)}, \quad [6.26]$$

3198 where $\beta_w \cdot v_u$ represents a dot product. As usual, the denominator ensures that the prob-
 3199 ability distribution is properly normalized. This vector of probabilities is equivalent to
 3200 applying the **softmax** transformation (see § 3.1) to the vector of dot-products,

$$p(\cdot | u) = \text{SoftMax}([\beta_1 \cdot v_u, \beta_2 \cdot v_u, \dots, \beta_V \cdot v_u]). \quad [6.27]$$

The word vectors β_w are parameters of the model, and are estimated directly. The context vectors v_u can be computed in various ways, depending on the model. A simple

²This idea predates neural language models (e.g., Rosenfeld, 1996; Roark et al., 2007).

but effective neural language model can be built from a **recurrent neural network** (RNN; Mikolov et al., 2010). The basic idea is to recurrently update the context vectors while moving through the sequence. Let \mathbf{h}_m represent the contextual information at position m in the sequence. RNN language models are defined,

$$\mathbf{x}_m \triangleq \phi_{w_m} \quad [6.28]$$

$$\mathbf{h}_m = \text{RNN}(\mathbf{x}_m, \mathbf{h}_{m-1}) \quad [6.29]$$

$$p(w_{m+1} | w_1, w_2, \dots, w_m) = \frac{\exp(\beta_{w_{m+1}} \cdot \mathbf{h}_m)}{\sum_{w' \in \mathcal{V}} \exp(\beta_{w'} \cdot \mathbf{h}_m)}, \quad [6.30]$$

where ϕ is a matrix of **input word embeddings**, and \mathbf{x}_m denotes the embedding for word w_m . The conversion of w_m to \mathbf{x}_m is sometimes known as a **lookup layer**, because we simply lookup the embeddings for each word in a table; see § 3.2.4.

The **Elman unit** defines a simple recurrent operation (Elman, 1990),

$$\text{RNN}(\mathbf{x}_m, \mathbf{h}_{m-1}) \triangleq g(\Theta \mathbf{h}_{m-1} + \mathbf{x}_m), \quad [6.31]$$

where $\Theta \in \mathbb{R}^{K \times K}$ is the recurrence matrix and g is a non-linear transformation function, often defined as the elementwise hyperbolic tangent \tanh (see § 3.1).³ The \tanh acts as a **squashing function**, ensuring that each element of \mathbf{h}_m is constrained to the range $[-1, 1]$.

Although each w_m depends on only the context vector \mathbf{h}_{m-1} , this vector is in turn influenced by *all* previous tokens, w_1, w_2, \dots, w_{m-1} , through the recurrence operation: w_1 affects \mathbf{h}_1 , which affects \mathbf{h}_2 , and so on, until the information is propagated all the way to \mathbf{h}_{m-1} , and then on to w_m (see Figure 6.1). This is an important distinction from n -gram language models, where any information outside the n -word window is ignored. In principle, the RNN language model can handle long-range dependencies, such as number agreement over long spans of text — although it would be difficult to know where exactly in the vector \mathbf{h}_m this information is represented. The main limitation is that information is attenuated by repeated application of the squashing function g . **Long short-term memories** (LSTMs), described below, are a variant of RNNs that address this issue, using memory cells to propagate information through the sequence without applying nonlinearities (Hochreiter and Schmidhuber, 1997).

The denominator in Equation 6.30 is a computational bottleneck, because it involves a sum over the entire vocabulary. One solution is to use a **hierarchical softmax** function, which computes the sum more efficiently by organizing the vocabulary into a tree (Mikolov et al., 2011). Another strategy is to optimize an alternative metric, such as **noise-contrastive estimation** (Gutmann and Hyvärinen, 2012), which learns by distinguishing observed instances from artificial instances generated from a noise distribution (Mnih and Teh, 2012). Both of these strategies are described in § 14.5.3.

³In the original Elman network, the sigmoid function was used in place of \tanh . For an illuminating mathematical discussion of the advantages and disadvantages of various nonlinearities in recurrent neural networks, see the lecture notes from Cho (2015).

3227 **6.3.1 Backpropagation through time**

3228 The recurrent neural network language model has the following parameters:

- 3229 • $\phi_i \in \mathbb{R}^K$, the “input” word vectors (these are sometimes called **word embeddings**,
3230 since each word is embedded in a K -dimensional space);
- 3231 • $\beta_i \in \mathbb{R}^K$, the “output” word vectors;
- 3232 • $\Theta \in \mathbb{R}^{K \times K}$, the recurrence operator;
- 3233 • \mathbf{h}_0 , the initial state.

3234 Each of these parameters can be estimated by formulating an objective function over the
3235 training corpus, $L(\mathbf{w})$, and then applying **backpropagation** to obtain gradients on the
3236 parameters from a minibatch of training examples (see § 3.3.1). Gradient-based updates
3237 can be computed from an online learning algorithm such as stochastic gradient descent
3238 (see § 2.5.2).

3239 The application of backpropagation to recurrent neural networks is known as **back-**
3240 **propagation through time**, because the gradients on units at time m depend in turn on the
3241 gradients of units at earlier times $n < m$. Let ℓ_{m+1} represent the negative log-likelihood
3242 of word $m + 1$,

$$\ell_{m+1} = -\log p(w_{m+1} | w_1, w_2, \dots, w_m). \quad [6.32]$$

We require the gradient of this loss with respect to each parameter, such as $\theta_{k,k'}$, an individual element in the recurrence matrix Θ . Since the loss depends on the parameters only through \mathbf{h}_m , we can apply the chain rule of differentiation,

$$\frac{\partial \ell_{m+1}}{\partial \theta_{k,k'}} = \frac{\partial \ell_{m+1}}{\partial \mathbf{h}_m} \frac{\partial \mathbf{h}_m}{\partial \theta_{k,k'}}. \quad [6.33]$$

The vector \mathbf{h}_m depends on Θ in several ways. First, \mathbf{h}_m is computed by multiplying Θ by the previous state \mathbf{h}_{m-1} . But the previous state \mathbf{h}_{m-1} also depends on Θ :

$$\mathbf{h}_m = g(\mathbf{x}_m, \mathbf{h}_{m-1}) \quad [6.34]$$

$$\frac{\partial h_{m,k}}{\partial \theta_{k,k'}} = g'(\mathbf{x}_{m,k} + \boldsymbol{\theta}_k \cdot \mathbf{h}_{m-1})(h_{m-1,k'} + \boldsymbol{\theta}_k \cdot \frac{\partial \mathbf{h}_{m-1}}{\partial \theta_{k,k'}}), \quad [6.35]$$

3243 where g' is the local derivative of the nonlinear function g . The key point in this equation
3244 is that the derivative $\frac{\partial \mathbf{h}_m}{\partial \theta_{k,k'}}$ depends on $\frac{\partial \mathbf{h}_{m-1}}{\partial \theta_{k,k'}}$, which will depend in turn on $\frac{\partial \mathbf{h}_{m-2}}{\partial \theta_{k,k'}}$, and
3245 so on, until reaching the initial state \mathbf{h}_0 .

3246 Each derivative $\frac{\partial \mathbf{h}_m}{\partial \theta_{k,k'}}$ will be reused many times: it appears in backpropagation from
3247 the loss ℓ_m , but also in all subsequent losses $\ell_{n>m}$. Neural network toolkits such as
3248 Torch (Collobert et al., 2011) and DyNet (Neubig et al., 2017) compute the necessary

3249 derivatives automatically, and cache them for future use. An important distinction from
3250 the feedforward neural networks considered in chapter 3 is that the size of the computa-
3251 tion graph is not fixed, but varies with the length of the input. This poses difficulties for
3252 toolkits that are designed around static computation graphs, such as TensorFlow (Abadi
3253 et al., 2016).⁴

3254 **6.3.2 Hyperparameters**

3255 The RNN language model has several hyperparameters that must be tuned to ensure good
3256 performance. The model capacity is controlled by the size of the word and context vectors
3257 K , which play a role that is somewhat analogous to the size of the n -gram context. For
3258 datasets that are large with respect to the vocabulary (i.e., there is a large token-to-type
3259 ratio), we can afford to estimate a model with a large K , which enables more subtle dis-
3260 tinctions between words and contexts. When the dataset is relatively small, then K must
3261 be smaller too, or else the model may “memorize” the training data, and fail to generalize.
3262 Unfortunately, this general advice has not yet been formalized into any concrete formula
3263 for choosing K , and trial-and-error is still necessary. Overfitting can also be prevented by
3264 **dropout**, which involves randomly setting some elements of the computation to zero (Sri-
3265 vastava et al., 2014), forcing the learner not to rely too much on any particular dimension
3266 of the word or context vectors. The dropout rate must also be tuned on development data.

3267 **6.3.3 Gated recurrent neural networks**

3268 In principle, recurrent neural networks can propagate information across infinitely long
3269 sequences. But in practice, repeated applications of the nonlinear recurrence function
3270 causes this information to be quickly attenuated. The same problem affects learning: back-
3271 propagation can lead to **vanishing gradients** that decay to zero, or **exploding gradients**
3272 that increase towards infinity (Bengio et al., 1994). The exploding gradient problem can
3273 be addressed by clipping gradients at some maximum value (Pascanu et al., 2013). The
3274 other issues must be addressed by altering the model itself.

3275 The **long short-term memory (LSTM)** (Hochreiter and Schmidhuber, 1997) is a popular
3276 variant of RNNs that is more robust to these problems. This model augments the hidden
3277 state \mathbf{h}_m with a **memory cell** c_m . The value of the memory cell at each time m is a gated
3278 sum of two quantities: its previous value c_{m-1} , and an “update” \tilde{c}_m , which is computed
3279 from the current input x_m and the previous hidden state \mathbf{h}_{m-1} . The next state \mathbf{h}_m is then
3280 computed from the memory cell. Because the memory cell is not passed through a non-
3281 linear squashing function during the update, it is possible for information to propagate
3282 through the network over long distances.

⁴See <https://www.tensorflow.org/tutorials/recurrent> (retrieved Feb 8, 2018).

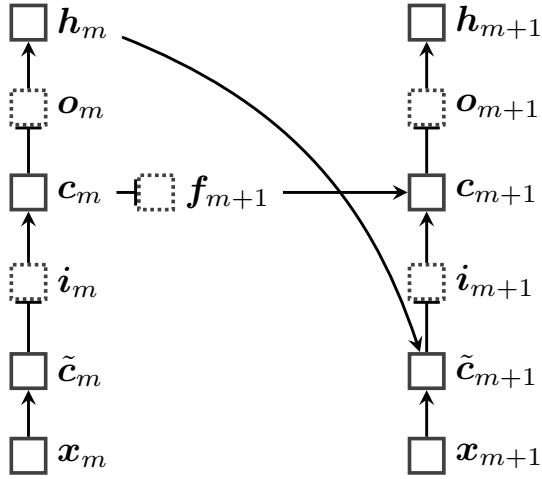


Figure 6.2: The long short-term memory (LSTM) architecture. Gates are shown in boxes with dotted edges. In an LSTM language model, each h_m would be used to predict the next word w_{m+1} .

The gates are functions of the input and previous hidden state. They are computed from elementwise sigmoid activations, $\sigma(x) = (1 + \exp(-x))^{-1}$, ensuring that their values will be in the range $[0, 1]$. They can therefore be viewed as soft, differentiable logic gates. The LSTM architecture is shown in Figure 6.2, and the complete update equations are:

$$f_{m+1} = \sigma(\Theta^{(h \rightarrow f)} h_m + \Theta^{(x \rightarrow f)} x_{m+1} + b_f) \quad \text{forget gate} \quad [6.36]$$

$$i_{m+1} = \sigma(\Theta^{(h \rightarrow i)} h_m + \Theta^{(x \rightarrow i)} x_{m+1} + b_i) \quad \text{input gate} \quad [6.37]$$

$$\tilde{c}_{m+1} = \tanh(\Theta^{(h \rightarrow c)} h_m + \Theta^{(x \rightarrow c)} x_{m+1}) \quad \text{update candidate} \quad [6.38]$$

$$c_{m+1} = f_{m+1} \odot c_m + i_{m+1} \odot \tilde{c}_{m+1} \quad \text{memory cell update} \quad [6.39]$$

$$o_{m+1} = \sigma(\Theta^{(h \rightarrow o)} h_m + \Theta^{(x \rightarrow o)} x_{m+1} + b_o) \quad \text{output gate} \quad [6.40]$$

$$h_{m+1} = o_{m+1} \odot \tanh(c_{m+1}) \quad \text{output.} \quad [6.41]$$

3283 The operator \odot is an elementwise (Hadamard) product. Each gate is controlled by a vec-
 3284 tor of weights, which parametrize the previous hidden state (e.g., $\Theta^{(h \rightarrow f)}$) and the current
 3285 input (e.g., $\Theta^{(x \rightarrow f)}$), plus a vector offset (e.g., b_f). The overall operation can be infor-
 3286 mally summarized as $(h_m, c_m) = \text{LSTM}(x_m, (h_{m-1}, c_{m-1}))$, with (h_m, c_m) representing
 3287 the LSTM state after reading token m .

3288 The LSTM outperforms standard recurrent neural networks across a wide range of
 3289 problems. It was first used for language modeling by Sundermeyer et al. (2012), but can
 3290 be applied more generally: the vector h_m can be treated as a complete representation of

3291 the input sequence up to position m , and can be used for any labeling task on a sequence
 3292 of tokens, as we will see in the next chapter.

3293 There are several LSTM variants, of which the Gated Recurrent Unit (Cho et al., 2014)
 3294 is one of the more well known. Many software packages implement a variety of RNN
 3295 architectures, so choosing between them is simple from a user’s perspective. Jozefowicz
 3296 et al. (2015) provide an empirical comparison of various modeling choices circa 2015.

3297 6.4 Evaluating language models

3298 Language modeling is not usually an application in itself: language models are typically
 3299 components of larger systems, and they would ideally be evaluated **extrinsically**. This
 3300 means evaluating whether the language model improves performance on the application
 3301 task, such as machine translation or speech recognition. But this is often hard to do, and
 3302 depends on details of the overall system which may be irrelevant to language modeling.
 3303 In contrast, **intrinsic evaluation** is task-neutral. Better performance on intrinsic metrics
 3304 may be expected to improve extrinsic metrics across a variety of tasks, but there is always
 3305 the risk of over-optimizing the intrinsic metric. This section discusses some intrinsic met-
 3306 rics, but keep in mind the importance of performing extrinsic evaluations to ensure that
 3307 intrinsic performance gains carry over to the applications that we care about.

3308 6.4.1 Held-out likelihood

The goal of probabilistic language models is to accurately measure the probability of sequences of word tokens. Therefore, an intrinsic evaluation metric is the likelihood that the language model assigns to **held-out data**, which is not used during training. Specifically, we compute,

$$\ell(\mathbf{w}) = \sum_{m=1}^M \log p(w_m | w_{m-1}, \dots, w_1), \quad [6.42]$$

3309 treating the entire held-out corpus as a single stream of tokens.

3310 Typically, unknown words are mapped to the $\langle \text{UNK} \rangle$ token. This means that we have
 3311 to estimate some probability for $\langle \text{UNK} \rangle$ on the training data. One way to do this is to fix
 3312 the vocabulary \mathcal{V} to the $V - 1$ words with the highest counts in the training data, and then
 3313 convert all other tokens to $\langle \text{UNK} \rangle$. Other strategies for dealing with out-of-vocabulary
 3314 terms are discussed in § 6.5.

3315 **6.4.2 Perplexity**

Held-out likelihood is usually presented as **perplexity**, which is a deterministic transformation of the log-likelihood into an information-theoretic quantity,

$$\text{Perplex}(\mathbf{w}) = 2^{-\frac{\ell(\mathbf{w})}{M}}, \quad [6.43]$$

3316 where M is the total number of tokens in the held-out corpus.

3317 Lower perplexities correspond to higher likelihoods, so lower scores are better on this
3318 metric — it is better to be less perplexed. Here are some special cases:

- 3319 • In the limit of a perfect language model, probability 1 is assigned to the held-out
3320 corpus, with $\text{Perplex}(\mathbf{w}) = 2^{-\frac{1}{M} \log_2 1} = 2^0 = 1$.
- 3321 • In the opposite limit, probability zero is assigned to the held-out corpus, which cor-
3322 responds to an infinite perplexity, $\text{Perplex}(\mathbf{w}) = 2^{-\frac{1}{M} \log_2 0} = 2^\infty = \infty$.
- 3323 • Assume a uniform, unigram model in which $p(w_i) = \frac{1}{V}$ for all words in the vocab-
3324 ular. Then,

$$\begin{aligned} \log_2(\mathbf{w}) &= \sum_{m=1}^M \log_2 \frac{1}{V} = - \sum_{m=1}^M \log_2 V = -M \log_2 V \\ \text{Perplex}(\mathbf{w}) &= 2^{\frac{1}{M} M \log_2 V} \\ &= 2^{\log_2 V} \\ &= V. \end{aligned}$$

3325 This is the “worst reasonable case” scenario, since you could build such a language
3326 model without even looking at the data.

3327 In practice, language models tend to give perplexities in the range between 1 and V .
3328 A small benchmark dataset is the **Penn Treebank**, which contains roughly a million to-
3329 kens; its vocabulary is limited to 10,000 words, with all other tokens mapped a special
3330 $\langle \text{UNK} \rangle$ symbol. On this dataset, a well-smoothed 5-gram model achieves a perplexity of
3331 141 (Mikolov and Zweig, Mikolov and Zweig), and an LSTM language model achieves
3332 perplexity of roughly 80 (Zaremba, Sutskever, and Vinyals, Zaremba et al.). Various en-
3333 hancements to the LSTM architecture can bring the perplexity below 60 (Merity et al.,
3334 2018). A larger-scale language modeling dataset is the 1B Word Benchmark (Chelba et al.,
3335 2013), which contains text from Wikipedia. On this dataset, a perplexities of around 25
can be obtained by averaging together multiple LSTM language models (Jozefowicz et al.,
2016).

3336 **6.5 Out-of-vocabulary words**

3337 So far, we have assumed a **closed-vocabulary** setting — the vocabulary \mathcal{V} is assumed to be
 3338 a finite set. In realistic application scenarios, this assumption may not hold. Consider, for
 3339 example, the problem of translating newspaper articles. The following sentence appeared
 3340 in a Reuters article on January 6, 2017:⁵

3341 The report said U.S. intelligence agencies believe Russian military intelligence,
 3342 the **GRU**, used intermediaries such as **WikiLeaks**, **DCLeaks.com** and the **Guc-**
 3343 **cifer** 2.0 "persona" to release emails...

3344 Suppose that you trained a language model on the Gigaword corpus,⁶ which was released
 3345 in 2003. The bolded terms either did not exist at this date, or were not widely known; they
 3346 are unlikely to be in the vocabulary. The same problem can occur for a variety of other
 3347 terms: new technologies, previously unknown individuals, new words (e.g., *hashtag*), and
 3348 numbers.

3349 One solution is to simply mark all such terms with a special token, $\langle \text{UNK} \rangle$. While
 3350 training the language model, we decide in advance on the vocabulary (often the K most
 3351 common terms), and mark all other terms in the training data as $\langle \text{UNK} \rangle$. If we do not want
 3352 to determine the vocabulary size in advance, an alternative approach is to simply mark
 3353 the first occurrence of each word type as $\langle \text{UNK} \rangle$.

3354 But it is often better to make distinctions about the likelihood of various unknown words.
 3355 This is particularly important in languages that have rich morphological systems, with
 3356 many inflections for each word. For example, Portuguese is only moderately complex
 3357 from a morphological perspective, yet each verb has dozens of inflected forms (see Fig-
 3358 ure 4.3b). In such languages, there will be many word types that we do not encounter in a
 3359 corpus, which are nonetheless predictable from the morphological rules of the language.
 3360 To use a somewhat contrived English example, if *transfenestrate* is in the vocabulary, our
 3361 language model should assign a non-zero probability to the past tense *transfenestrated*,
 3362 even if it does not appear in the training data.

3363 One way to accomplish this is to supplement word-level language models with **character-**
 3364 **level language models**. Such models can use n -grams or RNNs, but with a fixed vocab-
 3365 uary equal to the set of ASCII or Unicode characters. For example Ling et al. (2015)
 3366 propose an LSTM model over characters, and Kim (2014) employ a **convolutional neural**
 3367 **network** (LeCun and Bengio, 1995). A more linguistically motivated approach is to seg-
 3368 ment words into meaningful subword units, known as **morphemes** (see chapter 9). For

⁵Bayoumy, Y. and Strobel, W. (2017, January 6). U.S. intel report: Putin directed cy-
 ber campaign to help Trump. *Reuters*. Retrieved from <http://www.reuters.com/article/us-usa-russia-cyber-idUSKBN14Q1T8> on January 7, 2017.

⁶<https://catalog.ldc.upenn.edu/LDC2003T05>

3369 example, Botha and Blunsom (2014) induce vector representations for morphemes, which
3370 they build into a log-bilinear language model; Bhatia et al. (2016) incorporate morpheme
3371 vectors into an LSTM.

3372 Additional resources

3373 A variety of neural network architectures have been applied to language modeling. No-
3374 table earlier non-recurrent architectures include the neural probabilistic language model (Ben-
3375 gio et al., 2003) and the log-bilinear language model (Mnih and Hinton, 2007). Much more
3376 detail on these models can be found in the text by Goodfellow et al. (2016).

3377 Exercises

3378 1. exercises tk

3379 **Chapter 7**

3380 **Sequence labeling**

3381 The goal of sequence labeling is to assign tags to words, or more generally, to assign dis-
3382 crete labels to discrete elements in a sequence. There are many applications of sequence
3383 labeling in natural language processing, and chapter 8 presents an overview. A classic ap-
3384 plication is **part-of-speech tagging**, which involves tagging each word by its grammatical
3385 category. Coarse-grained grammatical categories include **NOUNs**, which describe things,
3386 properties, or ideas, and **VERBs**, which describe actions and events. Consider a simple
3387 input:

3388 (7.1) They can fish.

3389 A dictionary of coarse-grained part-of-speech tags might include **NOUN** as the only valid
3390 tag for *they*, but both **NOUN** and **VERB** as potential tags for *can* and *fish*. An accurate se-
3391 quence labeling algorithm should select the verb tag for both *can* and *fish* in (7.1), but it
3392 should select the noun tags for the same two words in the phrase *can of fish*.

3393 **7.1 Sequence labeling as classification**

One way to solve a tagging problem is to turn it into a classification problem. Let $f((\mathbf{w}, m), y)$ indicate the feature function for tag y at position m in the sequence $\mathbf{w} = (w_1, w_2, \dots, w_M)$. A simple tagging model would have a single base feature, the word itself:

$$f((\mathbf{w} = \text{they can fish}, m = 1), \text{N}) = (\text{they}, \text{N}) \quad [7.1]$$

$$f((\mathbf{w} = \text{they can fish}, m = 2), \text{V}) = (\text{can}, \text{V}) \quad [7.2]$$

$$f((\mathbf{w} = \text{they can fish}, m = 3), \text{V}) = (\text{fish}, \text{V}). \quad [7.3]$$

3394 Here the feature function takes three arguments as input: the sentence to be tagged (e.g.,
3395 *they can fish*), the proposed tag (e.g., N or V), and the index of the token to which this tag

3396 is applied. This simple feature function then returns a single feature: a tuple including
 3397 the word to be tagged and the tag that has been proposed. If the vocabulary size is V
 3398 and the number of tags is K , then there are $V \times K$ features. Each of these features must
 3399 be assigned a weight. These weights can be learned from a labeled dataset using a clas-
 3400 sification algorithm such as perceptron, but this isn't necessary in this case: it would be
 3401 equivalent to define the classification weights directly, with $\theta_{w,y} = 1$ for the tag y most
 3402 frequently associated with word w , and $\theta_{w,y} = 0$ for all other tags.

However, it is easy to see that this simple classification approach cannot correctly tag both *they can fish* and *can of fish*, because *can* and *fish* are grammatically ambiguous. To handle both of these cases, the tagger must rely on context, such as the surrounding words. We can build context into the feature set by incorporating the surrounding words as additional features:

$$\begin{aligned} f((\mathbf{w} = \text{they can fish}, 1), \mathbf{N}) = & \{(w_m = \text{they}, y_m = \mathbf{N}), \\ & (w_{m-1} = \square, y_m = \mathbf{N}), \\ & (w_{m+1} = \text{can}, y_m = \mathbf{N})\} \end{aligned} \quad [7.4]$$

$$\begin{aligned} f((\mathbf{w} = \text{they can fish}, 2), \mathbf{V}) = & \{(w_m = \text{can}, y_m = \mathbf{V}), \\ & (w_{m-1} = \text{they}, y_m = \mathbf{V}), \\ & (w_{m+1} = \text{fish}, y_m = \mathbf{V})\} \end{aligned} \quad [7.5]$$

$$\begin{aligned} f((\mathbf{w} = \text{they can fish}, 3), \mathbf{V}) = & \{(w_m = \text{fish}, y_m = \mathbf{V}), \\ & (w_{m-1} = \text{can}, y_m = \mathbf{V}), \\ & (w_{m+1} = \blacksquare, y_m = \mathbf{V})\}. \end{aligned} \quad [7.6]$$

3403 These features contain enough information that a tagger should be able to choose the
 3404 right tag for the word *fish*: words that come after *can* are likely to be verbs, so the feature
 3405 $(w_{m-1} = \text{can}, y_m = \mathbf{V})$ should have a large positive weight.

3406 However, even with this enhanced feature set, it may be difficult to tag some se-
 3407 quences correctly. One reason is that there are often relationships between the tags them-
 3408 selves. For example, in English it is relatively rare for a verb to follow another verb —
 3409 particularly if we differentiate MODAL verbs like *can* and *should* from more typical verbs,
 3410 like *give*, *transcend*, and *befuddle*. We would like to incorporate preferences against tag se-
 3411 quences like VERB-VERB, and in favor of tag sequences like NOUN-VERB. The need for
 3412 such preferences is best illustrated by a **garden path sentence**:

3413 (7.2) The old man the boat.

3414 Grammatically, the word *the* is a DETERMINER. When you read the sentence, what
 3415 part of speech did you first assign to *old*? Typically, this word is an ADJECTIVE — abbrevi-
 3416 ated as J — which is a class of words that modify nouns. Similarly, *man* is usually a noun.
 3417 The resulting sequence of tags is D J N D N. But this is a mistaken “garden path” inter-
 3418 pretation, which ends up leading nowhere. It is unlikely that a determiner would directly

follow a noun,¹ and it is particularly unlikely that the entire sentence would lack a verb. The only possible verb in (7.2) is the word *man*, which can refer to the act of maintaining and piloting something — often boats. But if *man* is tagged as a verb, then *old* is seated between a determiner and a verb, and must be a noun. And indeed, adjectives often have a second interpretation as nouns when used in this way (e.g., *the young*, *the restless*). This reasoning, in which the labeling decisions are intertwined, cannot be applied in a setting where each tag is produced by an independent classification decision.

7.2 Sequence labeling as structure prediction

As an alternative, think of the entire sequence of tags as a label itself. For a given sequence of words $\mathbf{w} = (w_1, w_2, \dots, w_M)$, there is a set of possible taggings $\mathcal{Y}(\mathbf{w}) = \mathcal{Y}^M$, where $\mathcal{Y} = \{\text{N, V, D, ...}\}$ refers to the set of individual tags, and \mathcal{Y}^M refers to the set of tag sequences of length M . We can then treat the sequence labeling problem as a classification problem in the label space $\mathcal{Y}(\mathbf{w})$,

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}(\mathbf{w})}{\operatorname{argmax}} \Psi(\mathbf{w}, \mathbf{y}), \quad [7.7]$$

where $\mathbf{y} = (y_1, y_2, \dots, y_M)$ is a sequence of M tags, and Ψ is a scoring function on pairs of sequences, $V^M \times \mathcal{Y}^M \mapsto \mathbb{R}$. Such a function can include features that capture the relationships between tagging decisions, such as the preference that determiners not follow nouns, or that all sentences have verbs.

Given that the label space is exponentially large in the length of the sequence M , can it ever be practical to perform tagging in this way? The problem of making a series of interconnected labeling decisions is known as **inference**. Because natural language is full of interrelated grammatical structures, inference is a crucial aspect of natural language processing. In English, it is not unusual to have sentences of length $M = 20$; part-of-speech tag sets vary in size from 10 to several hundred. Taking the low end of this range, we have $|\mathcal{Y}(\mathbf{w}_{1:M})| \approx 10^{20}$, one hundred billion billion possible tag sequences. Enumerating and scoring each of these sequences would require an amount of work that is exponential in the sequence length, so inference is intractable.

However, the situation changes when we restrict the scoring function. Suppose we choose a function that decomposes into a sum of local parts,

$$\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m), \quad [7.8]$$

where each $\psi(\cdot)$ scores a local part of the tag sequence. Note that the sum goes up to $M+1$, so that we can include a score for a special end-of-sequence tag, $\psi(\mathbf{w}_{1:M}, \diamond, y_M, M+1)$. We also define a special tag to begin the sequence, $y_0 \triangleq \diamond$.

¹The main exception occurs with ditransitive verbs, such as *They gave the winner a trophy*.

3448 In a linear model, local scoring function can be defined as a dot product of weights
 3449 and features,

$$\psi(\mathbf{w}_{1:M}, y_m, y_{m-1}, m) = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m). \quad [7.9]$$

3450 The feature vector \mathbf{f} can consider the entire input \mathbf{w} , and can look at pairs of adjacent
 3451 tags. This is a step up from per-token classification: the weights can assign low scores
 3452 to infelicitous tag pairs, such as noun-determiner, and high scores for frequent tag pairs,
 3453 such as determiner-noun and noun-verb.

In the example *they can fish*, a minimal feature function would include features for word-tag pairs (sometimes called **emission features**) and tag-tag pairs (sometimes called **transition features**):

$$\begin{aligned} \mathbf{f}(\mathbf{w} = \text{they can fish}, \mathbf{y} = \text{N V V}) &= \sum_{m=1}^{M+1} \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m) \\ &= \mathbf{f}(\mathbf{w}, \text{N}, \diamond, 1) \\ &\quad + \mathbf{f}(\mathbf{w}, \text{V}, \text{N}, 2) \\ &\quad + \mathbf{f}(\mathbf{w}, \text{V}, \text{V}, 3) \\ &\quad + \mathbf{f}(\mathbf{w}, \blacklozenge, \text{V}, 4) \end{aligned} \quad [7.10]$$

$$\begin{aligned} &= (w_m = \text{they}, y_m = \text{N}) + (y_m = \text{N}, y_{m-1} = \diamond) \\ &\quad + (w_m = \text{can}, y_m = \text{V}) + (y_m = \text{V}, y_{m-1} = \text{N}) \\ &\quad + (w_m = \text{fish}, y_m = \text{V}) + (y_m = \text{V}, y_{m-1} = \text{V}) \\ &\quad + (y_m = \blacklozenge, y_{m-1} = \text{V}). \end{aligned} \quad [7.11]$$

3454 There are seven active features for this example: one for each word-tag pair, and one
 3455 for each tag-tag pair, including a final tag $y_{M+1} = \blacklozenge$. These features capture the two main
 3456 sources of information for part-of-speech tagging in English: which tags are appropriate
 3457 for each word, and which tags tend to follow each other in sequence. Given appropriate
 3458 weights for these features, taggers can achieve high accuracy, even for difficult cases like
 3459 *the old man the boat*. We will now discuss how this restricted scoring function enables
 3460 efficient inference, through the **Viterbi algorithm** (Viterbi, 1967).

3461 **7.3 The Viterbi algorithm**

By decomposing the scoring function into a sum of local parts, it is possible to rewrite the tagging problem as follows:

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} \Psi(\mathbf{w}, \mathbf{y}) \quad [7.13]$$

$$= \operatorname{argmax}_{\mathbf{y}_{1:M}} \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.14]$$

$$= \operatorname{argmax}_{\mathbf{y}_{1:M}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}), \quad [7.15]$$

3462 where the final line simplifies the notation with the shorthand,

$$s_m(y_m, y_{m-1}) \triangleq \psi(\mathbf{w}_{1:M}, y_m, y_{m-1}, m). \quad [7.16]$$

This inference problem can be solved efficiently using **dynamic programming**, a algorithmic technique for reusing work in recurrent computations. As is often the case in dynamic programming, we begin by solving an auxiliary problem: rather than finding the best tag sequence, we simply compute the *score* of the best tag sequence,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}, \mathbf{y}_{1:M}) = \max_{\mathbf{y}_{1:M}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}). \quad [7.17]$$

This score involves a maximization over all tag sequences of length M , written $\max_{\mathbf{y}_{1:M}}$. This maximization can be broken into two pieces,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}, \mathbf{y}_{1:M}) = \max_{y_M} \max_{\mathbf{y}_{1:M-1}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}), \quad [7.18]$$

which simply says that we maximize over the final tag y_M , and we maximize over all “prefixes”, $\mathbf{y}_{1:M-1}$. But within the sum of scores, only the final term $s_{M+1}(\blacklozenge, y_M)$ depends on y_M . We can pull this term out of the second maximization,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}, \mathbf{y}_{1:M}) = \max_{y_M} s_{M+1}(\blacklozenge, y_M) + \max_{\mathbf{y}_{1:M-1}} \sum_{m=1}^M s_m(y_m, y_{m-1}). \quad [7.19]$$

This same reasoning can be applied recursively to the second term of Equation 7.19, pulling out $s_M(y_M, y_{M-1})$, and so on. We can formalize this idea by defining an auxiliary

Algorithm 11 The Viterbi algorithm. Each $s_m(k, k')$ is a local score for tag $y_m = k$ and $y_{m-1} = k'$.

```

for  $k \in \{0, \dots, K\}$  do
     $v_1(k) = s_1(k, \diamond)$ 
for  $m \in \{2, \dots, M\}$  do
    for  $k \in \{0, \dots, K\}$  do
         $v_m(k) = \max_{k'} s_m(k, k') + v_{m-1}(k')$ 
         $b_m(k) = \operatorname{argmax}_{k'} s_m(k, k') + v_{m-1}(k')$ 
     $y_M = \operatorname{argmax}_k s_{M+1}(\blacklozenge, k) + v_M(k)$ 
    for  $m \in \{M-1, \dots, 1\}$  do
         $y_m = b_m(y_{m+1})$ 
return  $\mathbf{y}_{1:M}$ 
```

Viterbi variable,

$$v_m(y_m) \triangleq \max_{\mathbf{y}_{1:m-1}} \sum_{n=1}^m s_n(y_n, y_{n-1}) \quad [7.20]$$

$$= \max_{y_{m-1}} s_m(y_m, y_{m-1}) + \max_{\mathbf{y}_{1:m-2}} \sum_{n=1}^{m-1} s_n(y_n, y_{n-1}) \quad [7.21]$$

$$= \max_{y_{m-1}} s_m(y_m, y_{m-1}) + v_{m-1}(y_{m-1}). \quad [7.22]$$

3463 The variable $v_m(k)$ represents the score of the best sequence of length m ending in tag k .

Each set of Viterbi variables is computed from the local score $s_m(y_m, y_{m-1})$, and from the previous set of Viterbi variables. The initial condition of the recurrence is simply the first score,

$$v_1(y_1) \triangleq s_1(y_1, \diamond). \quad [7.23]$$

The maximum overall score for the sequence is then the final Viterbi variable,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}_{1:M}, \mathbf{y}_{1:M}) = v_{M+1}(\blacklozenge). \quad [7.24]$$

3464 Thus, the score of the best labeling for the sequence can be computed in a single forward
 3465 sweep: first compute all variables $v_1(\cdot)$ from Equation 7.23, and then compute all variables
 3466 $v_2(\cdot)$ from the recurrence Equation 7.22, and continue until reaching the final variable
 3467 $v_{M+1}(\blacklozenge)$.

3468 Graphically, it is customary to arrange these variables in a structure known as a **trellis**,
 3469 shown in Figure 7.1. Each column indexes a token m in the sequence, and each row

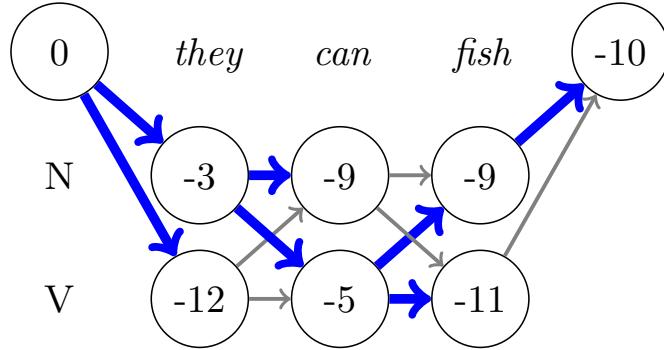


Figure 7.1: The trellis representation of the Viterbi variables, for the example *they can fish*, using the weights shown in Table 7.1.

3470 indexes a tag in \mathcal{Y} ; every $v_{m-1}(k)$ is connected to every $v_m(k')$, that $v_m(k')$ is computed
 3471 from $v_{m-1}(k)$. Special nodes are set aside for the start and end states.

3472 Our real goal is to find the best scoring sequence, not simply to compute its score.
 3473 But solving the auxiliary problem gets us almost all the way there. Recall that each $v_m(k)$
 3474 represents the score of the best tag sequence ending in that tag k in position m . To compute
 3475 this, we maximize over possible values of y_{m-1} . If we keep track of the “argmax” tag that
 3476 maximizes this choice at each step, then we can walk backwards from the final tag, and
 3477 recover the optimal tag sequence. This is indicated in Figure 7.1 by the solid blue lines,
 3478 which we trace back from the final position. These “back-pointers” are written $b_m(k)$,
 3479 indicating the optimal tag y_{m-1} on the path to $Y_m = k$.

3480 The complete Viterbi algorithm is shown in Algorithm 11. When computing the initial
 3481 Viterbi variables $v_1(\cdot)$, we use a special tag, \diamond , to indicate the start of the sequence. When
 3482 computing the final tag Y_M , we use another special tag, \blacklozenge , to indicate the end of the
 3483 sequence. Linguistically, these special tags enable the use of transition features for the tags
 3484 that begin and end the sequence: for example, conjunctions are unlikely to end sentences
 3485 in English, so we would like a low score for $s_{M+1}(\blacklozenge, CC)$; nouns are relatively likely to
 3486 appear at the beginning of sentences, so we would like a high score for $s_1(N, \diamond)$, assuming
 3487 the noun tag is compatible with the first word token w_1 .

3488 **Complexity** If there are K tags and M positions in the sequence, then there are $M \times K$
 3489 Viterbi variables to compute. Computing each variable requires finding a maximum over
 3490 K possible predecessor tags. The total time complexity of populating the trellis is therefore
 3491 $\mathcal{O}(MK^2)$, with an additional factor for the number of active features at each position.
 3492 After completing the trellis, we simply trace the backwards pointers to the beginning of
 3493 the sequence, which takes $\mathcal{O}(M)$ operations.

	<i>they</i>	<i>can</i>	<i>fish</i>	
N	-2	-3	-3	
V	-10	-1	-3	

(a) Weights for emission features.

	N	V	♦
◊	-1	-2	$-\infty$
N	-3	-1	-1
V	-1	-3	-1

(b) Weights for transition features. The “from” tags are on the columns, and the “to” tags are on the rows.

Table 7.1: Feature weights for the example trellis shown in Figure 7.1. Emission weights from \diamond and ♦ are implicitly set to $-\infty$.3494

7.3.1 Example

3495 Consider the minimal tagset $\{N, V\}$, corresponding to nouns and verbs. Even in this
 3496 tagset, there is considerable ambiguity: for example, the words *can* and *fish* can each take
 3497 both tags. Of the $2 \times 2 \times 2 = 8$ possible taggings for the sentence *they can fish*, four are
 3498 possible given these possible tags, and two are grammatical.²

3499 The values in the trellis in Figure 7.1 are computed from the feature weights defined in
 3500 Table 7.1. We begin with $v_1(N)$, which has only one possible predecessor, the start tag \diamond .
 3501 This score is therefore equal to $s_1(N, \diamond) = -2 - 1 = -3$, which is the sum of the scores for
 3502 the emission and transition features respectively; the backpointer is $b_1(N) = \diamond$. The score
 3503 for $v_1(V)$ is computed in the same way: $s_1(V, \diamond) = -10 - 2 = -12$, and again $b_1(V) = \diamond$.
 3504 The backpointers are represented in the figure by thick lines.

Things get more interesting at $m = 2$. The score $v_2(N)$ is computed by maximizing over the two possible predecessors,

$$v_2(N) = \max(v_1(N) + s_2(N, N), v_1(V) + s_2(N, V)) \quad [7.25]$$

$$= \max(-3 - 3 - 3, -12 - 3 - 1) = -9 \quad [7.26]$$

$$b_2(N) = N. \quad [7.27]$$

This continues until reaching $v_4(\diamond)$, which is computed as,

$$v_4(\diamond) = \max(v_3(N) + s_4(\diamond, N), v_3(V) + s_4(\diamond, V)) \quad [7.28]$$

$$= \max(-9 + 0 - 1, -11 + 0 - 1) \quad [7.29]$$

$$= -10, \quad [7.30]$$

3505 so $b_4(\diamond) = N$. As there is no emission w_4 , the emission features have scores of zero.

²The tagging *they/N can/V fish/N* corresponds to the scenario of putting fish into cans, or perhaps of firing them.

3506 To compute the optimal tag sequence, we walk backwards from here, next checking
 3507 $b_3(N) = V$, and then $b_2(V) = N$, and finally $b_1(N) = \diamond$. This yields $\mathbf{y} = (N, V, N)$, which
 3508 corresponds to the linguistic interpretation of the fishes being put into cans.

3509 **7.3.2 Higher-order features**

3510 The Viterbi algorithm was made possible by a restriction of the scoring function to local
 3511 parts that consider only pairs of adjacent tags. We can think of this as a bigram language
 3512 model over tags. A natural question is how to generalize Viterbi to tag trigrams, which
 3513 would involve the following decomposition:

$$\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+2} \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, y_{m-2}, m), \quad [7.31]$$

3514 where $y_{-1} = \diamond$ and $y_{M+2} = \blacklozenge$.

3515 One solution is to create a new tagset $\mathcal{Y}^{(2)}$ from the Cartesian product of the original
 3516 tagset with itself, $\mathcal{Y}^{(2)} = \mathcal{Y} \times \mathcal{Y}$. The tags in this product space are ordered pairs, rep-
 3517 resenting adjacent tags at the token level: for example, the tag (N, V) would represent a
 3518 noun followed by a verb. Transitions between such tags must be consistent: we can have a
 3519 transition from (N, V) to (V, N) (corresponding to the tag sequence $N V N$), but not from
 3520 (N, V) to (N, N) , which would not correspond to any coherent tag sequence. This con-
 3521 straint can be enforced in feature weights, with $\theta_{((a,b),(c,d))} = -\infty$ if $b \neq c$. The remaining
 3522 feature weights can encode preferences for and against various tag trigrams.

3523 In the Cartesian product tag space, there are K^2 tags, suggesting that the time com-
 3524 plexity will increase to $\mathcal{O}(MK^4)$. However, it is unnecessary to max over predecessor tag
 3525 bigrams that are incompatible with the current tag bigram. By exploiting this constraint,
 3526 it is possible to limit the time complexity to $\mathcal{O}(MK^3)$. The space complexity grows to
 3527 $\mathcal{O}(MK^2)$, since the trellis must store all possible predecessors of each tag. In general, the
 3528 time and space complexity of higher-order Viterbi grows exponentially with the order of
 3529 the tag n -grams that are considered in the feature decomposition.

3530 **7.4 Hidden Markov Models**

3531 Let us now consider how to learn the scores $s_m(y, y')$ that parametrize the Viterbi sequence
 3532 labeling algorithm, beginning with a probabilistic approach. Recall from § 2.1 that the
 3533 probabilistic Naïve Bayes classifier selects the label y to maximize $p(y | \mathbf{x}) \propto p(y, \mathbf{x})$. In
 3534 probabilistic sequence labeling, our goal is similar: select the tag sequence that maximizes
 3535 $p(\mathbf{y} | \mathbf{w}) \propto p(\mathbf{y}, \mathbf{w})$. The locality restriction in Equation 7.8 can be viewed as a conditional
 3536 independence assumption on the random variables \mathbf{y} .

Algorithm 12 Generative process for the hidden Markov model

```

 $y_0 \leftarrow \diamond,$     $m \leftarrow 1$ 
repeat
     $y_m \sim \text{Categorical}(\lambda_{y_{m-1}})$             $\triangleright$  sample the current tag
     $w_m \sim \text{Categorical}(\phi_{y_m})$             $\triangleright$  sample the current word
until  $y_m = \blacklozenge$             $\triangleright$  terminate when the stop symbol is generated

```

3537 Naïve Bayes was introduced as a generative model — a probabilistic story that ex-
 3538 plains the observed data as well as the hidden label. A similar story can be constructed
 3539 for probabilistic sequence labeling: first, the tags are drawn from a prior distribution; next,
 3540 the tokens are drawn from a conditional likelihood. However, for inference to be tractable,
 3541 additional independence assumptions are required. First, the probability of each token
 3542 depends only on its tag, and not on any other element in the sequence:

$$p(w | y) = \prod_{m=1}^M p(w_m | y_m). \quad [7.32]$$

3543 Second, each tag y_m depends only on its predecessor,

$$p(y) = \prod_{m=1}^M p(y_m | y_{m-1}), \quad [7.33]$$

3544 where $y_0 = \diamond$ in all cases. Due to this **Markov assumption**, probabilistic sequence labeling
 3545 models are known as **hidden Markov models** (HMMs).

3546 The generative process for the hidden Markov model is shown in Algorithm 12. Given
 3547 the parameters λ and ϕ , we can compute $p(w, y)$ for any token sequence w and tag se-
 3548 quence y . The HMM is often represented as a **graphical model** (Wainwright and Jordan,
 3549 2008), as shown in Figure 7.2. This representation makes the independence assumptions
 3550 explicit: if a variable v_1 is probabilistically conditioned on another variable v_2 , then there
 3551 is an arrow $v_2 \rightarrow v_1$ in the diagram. If there are no arrows between v_1 and v_2 , they
 3552 are **conditionally independent**, given each variable's **Markov blanket**. In the hidden
 3553 Markov model, the Markov blanket for each tag y_m includes the “parent” y_{m-1} , and the
 3554 “children” y_{m+1} and w_m .³

3555 It is important to reflect on the implications of the HMM independence assumptions.
 3556 A non-adjacent pair of tags y_m and y_n are conditionally independent; if $m < n$ and we
 3557 are given y_{n-1} , then y_m offers no additional information about y_n . However, if we are
 3558 not given any information about the tags in a sequence, then all tags are probabilistically
 3559 coupled.

³In general graphical models, a variable's Markov blanket includes its parents, children, and its children's other parents (Murphy, 2012).

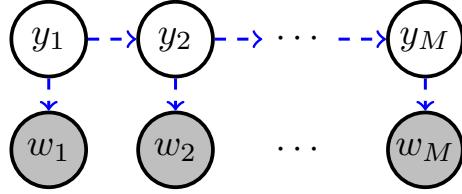


Figure 7.2: Graphical representation of the hidden Markov model. Arrows indicate probabilistic dependencies.

3560 7.4.1 Estimation

3561 The hidden Markov model has two groups of parameters:

3562 **Emission probabilities.** The probability $p_e(w_m | y_m; \phi)$ is the emission probability, since
3563 the words are treated as probabilistically “emitted”, conditioned on the tags.

3564 **Transition probabilities.** The probability $p_t(y_m | y_{m-1}; \lambda)$ is the transition probability,
3565 since it assigns probability to each possible tag-to-tag transition.

Both of these groups of parameters are typically computed from smoothed relative frequency estimation on a labeled corpus (see § 6.2 for a review of smoothing). The unsmoothed probabilities are,

$$\begin{aligned}\phi_{k,i} &\triangleq \Pr(W_m = i | Y_m = k) = \frac{\text{count}(W_m = i, Y_m = k)}{\text{count}(Y_m = k)} \\ \lambda_{k,k'} &\triangleq \Pr(Y_m = k' | Y_{m-1} = k) = \frac{\text{count}(Y_m = k', Y_{m-1} = k)}{\text{count}(Y_{m-1} = k)}.\end{aligned}$$

3566 Smoothing is more important for the emission probability than the transition probability,
3567 because the vocabulary is much larger than the number of tags.

3568 7.4.2 Inference

3569 The goal of inference in the hidden Markov model is to find the highest probability tag
3570 sequence,

$$\hat{y} = \underset{y}{\operatorname{argmax}} p(y | w). \quad [7.34]$$

3571 As in Naïve Bayes, it is equivalent to find the tag sequence with the highest *log*-probability,
3572 since the logarithm is a monotonically increasing function. It is furthermore equivalent
3573 to maximize the joint probability $p(y, w) = p(y | w) \times p(w) \propto p(y | w)$, which is pro-
3574 portional to the conditional probability. Putting these observations together, the inference

3575 problem can be reformulated as,

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y}} \log p(\mathbf{y}, \mathbf{w}). \quad [7.35]$$

We can now apply the HMM independence assumptions:

$$\log p(\mathbf{y}, \mathbf{w}) = \log p(\mathbf{y}) + \log p(\mathbf{w} \mid \mathbf{y}) \quad [7.36]$$

$$= \sum_{m=1}^{M+1} \log p_Y(y_m \mid y_{m-1}) + \log p_{W|Y}(w_m \mid y_m) \quad [7.37]$$

$$= \sum_{m=1}^{M+1} \log \lambda_{y_m, y_{m-1}} + \log \phi_{y_m, w_m} \quad [7.38]$$

$$= \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}), \quad [7.39]$$

where,

$$s_m(y_m, y_{m-1}) \triangleq \log \lambda_{y_m, y_{m-1}} + \log \phi_{y_m, w_m}, \quad [7.40]$$

3576 and,

$$\phi_{\diamond, w} = \begin{cases} 1, & w = \blacksquare \\ 0, & \text{otherwise,} \end{cases} \quad [7.41]$$

3577 which ensures that the stop tag \diamond can only be applied to the final token \blacksquare .

This derivation shows that HMM inference can be viewed as an application of the Viterbi decoding algorithm, given an appropriately defined scoring function. The local score $s_m(y_m, y_{m-1})$ can be interpreted probabilistically,

$$s_m(y_m, y_{m-1}) = \log p_y(y_m \mid y_{m-1}) + \log p_{w|y}(w_m \mid y_m) \quad [7.42]$$

$$= \log p(y_m, w_m \mid y_{m-1}). \quad [7.43]$$

Now recall the definition of the Viterbi variables,

$$v_m(y_m) = \max_{y_{m-1}} s_m(y_m, y_{m-1}) + v_{m-1}(y_{m-1}) \quad [7.44]$$

$$= \max_{y_{m-1}} \log p(y_m, w_m \mid y_{m-1}) + v_{m-1}(y_{m-1}). \quad [7.45]$$

By setting $v_{m-1}(y_{m-1}) = \max_{\mathbf{y}_{1:m-2}} \log p(\mathbf{y}_{1:m-1}, \mathbf{w}_{1:m-1})$, we obtain the recurrence,

$$v_m(y_m) = \max_{y_{m-1}} \log p(y_m, w_m \mid y_{m-1}) + \max_{\mathbf{y}_{1:m-2}} \log p(\mathbf{y}_{1:m-1}, \mathbf{w}_{1:m-1}) \quad [7.46]$$

$$= \max_{\mathbf{y}_{1:m-1}} \log p(y_m, w_m \mid y_{m-1}) + \log p(\mathbf{y}_{1:m-1}, \mathbf{w}_{1:m-1}) \quad [7.47]$$

$$= \max_{\mathbf{y}_{1:m-1}} \log p(\mathbf{y}_{1:m}, \mathbf{w}_{1:m}). \quad [7.48]$$

In words, the Viterbi variable $v_m(y_m)$ is the log probability of the best tag sequence ending in y_m , joint with the word sequence $w_{1:m}$. The log probability of the best complete tag sequence is therefore,

$$\max_{\mathbf{y}_{1:M}} \log p(\mathbf{y}_{1:M+1}, \mathbf{w}_{1:M+1}) = v_{M+1}(\spadesuit) \quad [7.49]$$

***Viterbi as an example of the max-product algorithm** The Viterbi algorithm can also be implemented using probabilities, rather than log-probabilities. In this case, each $v_m(y_m)$ is equal to,

$$v_m(y_m) = \max_{\mathbf{y}_{1:m-1}} p(\mathbf{y}_{1:m-1}, y_m, \mathbf{w}_{1:m}) \quad [7.50]$$

$$= \max_{y_{m-1}} p(y_m, w_m | y_{m-1}) \times \max_{\mathbf{y}_{1:m-2}} p(\mathbf{y}_{1:m-2}, y_{m-1}, \mathbf{w}_{1:m-1}) \quad [7.51]$$

$$= \max_{y_{m-1}} p(y_m, w_m | y_{m-1}) \times v_{m-1}(y_{m-1}) \quad [7.52]$$

$$= p_{w|y}(w_m | y_m) \times \max_{y_{m-1}} p_y(y_m | y_{m-1}) \times v_{m-1}(y_{m-1}). \quad [7.53]$$

3578 Each Viterbi variable is computed by *maximizing* over a set of *products*. Thus, the Viterbi
 3579 algorithm is a special case of the **max-product algorithm** for inference in graphical mod-
 3580 els (Wainwright and Jordan, 2008). However, the product of probabilities tends towards
 3581 zero over long sequences, so the log-probability version of Viterbi is recommended in
 3582 practical implementations.

3583 7.5 Discriminative sequence labeling with features

3584 Today, hidden Markov models are rarely used for supervised sequence labeling. This is
 3585 because HMMs are limited to only two phenomena:

- 3586 • word-tag compatibility, via the emission probability $p_{W|Y}(w_m | y_m)$;
- 3587 • local context, via the transition probability $p_Y(y_m | y_{m-1})$.

3588 The Viterbi algorithm permits the inclusion of richer information in the local scoring func-
 3589 tion $\psi(\mathbf{w}_{1:M}, y_m, y_{m-1}, m)$, which can be defined as a weighted sum of arbitrary local *fea-*
 3590 *tures*,

$$\psi(\mathbf{w}, y_m, y_{m-1}, m) = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m), \quad [7.54]$$

3591 where \mathbf{f} is a locally-defined feature function, and $\boldsymbol{\theta}$ is a vector of weights.

The local decomposition of the scoring function Ψ is reflected in a corresponding decomposition of the feature function:

$$\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.55]$$

$$= \sum_{m=1}^{M+1} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.56]$$

$$= \boldsymbol{\theta} \cdot \sum_{m=1}^{M+1} \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.57]$$

$$= \boldsymbol{\theta} \cdot \mathbf{f}^{(\text{global})}(\mathbf{w}, \mathbf{y}_{1:M}), \quad [7.58]$$

3592 where $\mathbf{f}^{(\text{global})}(\mathbf{w}, \mathbf{y})$ is a global feature vector, which is a sum of local feature vectors,

$$\mathbf{f}^{(\text{global})}(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \mathbf{f}(\mathbf{w}_{1:M}, y_m, y_{m-1}, m), \quad [7.59]$$

3593 with $y_{M+1} = \diamond$ and $y_0 = \diamond$ by construction.

3594 Let's now consider what additional information these features might encode.

3595 **Word affix features.** Consider the problem of part-of-speech tagging on the first four
3596 lines of the poem *Jabberwocky* (Carroll, 1917):

3597 (7.3) 'Twas brillig, and the slithy toves
3598 Did gyre and gimble in the wabe:
3599 All mimsy were the borogoves,
3600 And the mome raths outgrabe.

3601 Many of these words were made up by the author of the poem, so a corpus would offer
3602 no information about their probabilities of being associated with any particular part of
3603 speech. Yet it is not so hard to see what their grammatical roles might be in this passage.
3604 Context helps: for example, the word *slithy* follows the determiner *the*, so it is probably a
3605 noun or adjective. Which do you think is more likely? The suffix *-thy* is found in a number
3606 of adjectives, like *frothy*, *healthy*, *pithy*, *worthy*. It is also found in a handful of nouns — e.g.,
3607 *apathy*, *sympathy* — but nearly all of these have the longer coda *-pathy*, unlike *slithy*. So the
3608 suffix gives some evidence that *slithy* is an adjective, and indeed it is: later in the text we
3609 find that it is a combination of the adjectives *lithe* and *slimy*.⁴

⁴Morphology is the study of how words are formed from smaller linguistic units. Computational approaches to morphological analysis are touched on in chapter 9; Bender (2013) provides a good overview of the underlying linguistic principles.

3610 **Fine-grained context.** The hidden Markov model captures contextual information in the
 3611 form of part-of-speech tag bigrams. But sometimes, the necessary contextual information
 3612 is more specific. Consider the noun phrases *this fish* and *these fish*. Many part-of-speech
 3613 tagsets distinguish between singular and plural nouns, but do not distinguish between
 3614 singular and plural determiners.⁵ A hidden Markov model would be unable to correctly
 3615 label *fish* as singular or plural in both of these cases, because it only has access to two
 3616 features: the preceding tag (determiner in both cases) and the word (*fish* in both cases).
 3617 The classification-based tagger discussed in § 7.1 had the ability to use preceding and suc-
 3618 ceeding words as features, and it can also be incorporated into a Viterbi-based sequence
 3619 labeler as a local feature.

Example Consider the tagging D J N (determiner, adjective, noun) for the sequence *the slithy toves*, so that

$$\mathbf{w} = \text{the slithy toves}$$

$$\mathbf{y} = \text{D J N}.$$

Let's create the feature vector for this example, assuming that we have word-tag features (indicated by W), tag-tag features (indicated by T), and suffix features (indicated by M). You can assume that you have access to a method for extracting the suffix *-thy* from *slithy*, *-es* from *toves*, and \emptyset from *the*, indicating that this word has no suffix.⁶ The resulting feature vector is,

$$\begin{aligned} f(\text{the slithy toves, D J N}) &= f(\text{the slithy toves, D}, \diamond, 1) \\ &\quad + f(\text{the slithy toves, J}, D, 2) \\ &\quad + f(\text{the slithy toves, N}, J, 3) \\ &\quad + f(\text{the slithy toves, } \blacklozenge, N, 4) \\ &= \{(T : \diamond, D), (W : \text{the}, D), (M : \emptyset, D), \\ &\quad (T : D, J), (W : \text{slithy}, J), (M : \text{-thy}, J), \\ &\quad (T : J, N), (W : \text{toves}, N), (M : \text{-es}, N) \\ &\quad (T : N, \blacklozenge)\}. \end{aligned}$$

3620 These examples show that local features can incorporate information that lies beyond
 3621 the scope of a hidden Markov model. Because the features are local, it is possible to apply
 3622 the Viterbi algorithm to identify the optimal sequence of tags. The remaining question

⁵For example, the Penn Treebank tagset follows these conventions.

⁶Such a system is called a **morphological segmenter**. The task of morphological segmentation is briefly described in § 9.1.4.4; a well known segmenter is Morfessor (Creutz and Lagus, 2007). In real applications, a typical approach is to include features for all orthographic suffixes up to some maximum number of characters: for *slithy*, we would have suffix features for *-y*, *-hy*, and *-thy*.

3623 is how to estimate the weights on these features. § 2.2 presented three main types of
 3624 discriminative classifiers: perceptron, support vector machine, and logistic regression.
 3625 Each of these classifiers has a structured equivalent, enabling it to be trained from labeled
 3626 sequences rather than individual tokens.

3627 **7.5.1 Structured perceptron**

The perceptron classifier is trained by increasing the weights for features that are associated with the correct label, and decreasing the weights for features that are associated with incorrectly predicted labels:

$$\hat{y} = \operatorname{argmax}_{y \in \mathcal{Y}} \theta \cdot f(\mathbf{x}, y) \quad [7.60]$$

$$\theta^{(t+1)} \leftarrow \theta^{(t)} + f(\mathbf{x}, y) - f(\mathbf{x}, \hat{y}). \quad [7.61]$$

We can apply exactly the same update in the case of structure prediction,

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} \theta \cdot f(\mathbf{w}, \mathbf{y}) \quad [7.62]$$

$$\theta^{(t+1)} \leftarrow \theta^{(t)} + f(\mathbf{w}, \mathbf{y}) - f(\mathbf{w}, \hat{\mathbf{y}}). \quad [7.63]$$

3628 This learning algorithm is called **structured perceptron**, because it learns to predict the
 3629 structured output \mathbf{y} . The only difference is that instead of computing \hat{y} by enumerating
 3630 the entire set \mathcal{Y} , the Viterbi algorithm is used to efficiently search the set of possible tag-
 3631 gings, \mathcal{Y}^M . Structured perceptron can be applied to other structured outputs as long as
 3632 efficient inference is possible. As in perceptron classification, weight averaging is crucial
 3633 to get good performance (see § 2.2.2).

Example For the example *they can fish*, suppose that the reference tag sequence is $\mathbf{y}^{(i)} =$
 N V V, but the tagger incorrectly returns the tag sequence $\hat{\mathbf{y}} = \text{N V N}$. Assuming a model
 with features for emissions (w_m, y_m) and transitions (y_{m-1}, y_m) , the corresponding structured
 perceptron update is:

$$\theta_{(\text{fish}, \text{V})} \leftarrow \theta_{(\text{fish}, \text{V})} + 1, \quad \theta_{(\text{fish}, \text{N})} \leftarrow \theta_{(\text{fish}, \text{N})} - 1 \quad [7.64]$$

$$\theta_{(\text{V}, \text{V})} \leftarrow \theta_{(\text{V}, \text{V})} + 1, \quad \theta_{(\text{V}, \text{N})} \leftarrow \theta_{(\text{V}, \text{N})} - 1 \quad [7.65]$$

$$\theta_{(\text{V}, \blacklozenge)} \leftarrow \theta_{(\text{V}, \blacklozenge)} + 1, \quad \theta_{(\text{N}, \blacklozenge)} \leftarrow \theta_{(\text{N}, \blacklozenge)} - 1. \quad [7.66]$$

3634 **7.5.2 Structured support vector machines**

3635 Large-margin classifiers such as the support vector machine improve on the perceptron by
 3636 pushing the classification boundary away from the training instances. The same idea can

be applied to sequence labeling. A support vector machine in which the output is a structured object, such as a sequence, is called a **structured support vector machine** (Tsochan-taridis et al., 2004).⁷

In classification, we formalized the large-margin constraint as,

$$\forall \mathbf{y} \neq \mathbf{y}^{(i)}, \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}^{(i)}) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}) \geq 1, \quad [7.67]$$

requiring a margin of at least 1 between the scores for all labels \mathbf{y} that are not equal to the correct label $\mathbf{y}^{(i)}$. The weights $\boldsymbol{\theta}$ are then learned by constrained optimization (see § 2.3.2).

This idea can be applied to sequence labeling by formulating an equivalent set of constraints for all possible labelings $\mathcal{Y}(\mathbf{w})$ for an input \mathbf{w} . However, there are two problems. First, in sequence labeling, some predictions are more wrong than others: we may miss only one tag out of fifty, or we may get all fifty wrong. We would like our learning algorithm to be sensitive to this difference. Second, the number of constraints is equal to the number of possible labelings, which is exponentially large in the length of the sequence.

The first problem can be addressed by adjusting the constraint to require larger margins for more serious errors. Let $c(\mathbf{y}^{(i)}, \hat{\mathbf{y}}) \geq 0$ represent the *cost* of predicting label $\hat{\mathbf{y}}$ when the true label is $\mathbf{y}^{(i)}$. We can then generalize the margin constraint,

$$\forall \mathbf{y}, \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) \geq c(\mathbf{y}^{(i)}, \mathbf{y}). \quad [7.68]$$

This cost-augmented margin constraint specializes to the constraint in Equation 7.67 if we choose the delta function $c(\mathbf{y}^{(i)}, \mathbf{y}) = \delta((\mathbf{y}^{(i)} \neq \mathbf{y}))$. A more expressive cost function is the **Hamming cost**,

$$c(\mathbf{y}^{(i)}, \mathbf{y}) = \sum_{m=1}^M \delta(y_m^{(i)} \neq y_m), \quad [7.69]$$

which computes the number of errors in \mathbf{y} . By incorporating the cost function as the margin constraint, we require that the true labeling be separated from the alternatives by a margin that is proportional to the number of incorrect tags in each alternative labeling.

The second problem is that the number of constraints is exponential in the length of the sequence. This can be addressed by focusing on the prediction $\hat{\mathbf{y}}$ that *maximally* violates the margin constraint. This prediction can be identified by solving the following **cost-augmented decoding** problem:

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \neq \mathbf{y}^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) + c(\mathbf{y}^{(i)}, \mathbf{y}) \quad [7.70]$$

$$= \operatorname{argmax}_{\mathbf{y} \neq \mathbf{y}^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) + c(\mathbf{y}^{(i)}, \mathbf{y}), \quad [7.71]$$

⁷This model is also known as a **max-margin Markov network** (Taskar et al., 2003), emphasizing that the scoring function is constructed from a sum of components, which are Markov independent.

3658 where in the second line we drop the term $\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})$, which is constant in \mathbf{y} .

We can now reformulate the margin constraint for sequence labeling,

$$\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) - \max_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} (\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}) + c(\mathbf{y}^{(i)}, \mathbf{y})) \geq 0. \quad [7.72]$$

3659 If the score for $\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})$ is greater than the cost-augmented score for all alternatives,
 3660 then the constraint will be met. The name “cost-augmented decoding” is due to the fact
 3661 that the objective includes the standard decoding problem, $\max_{\hat{\mathbf{y}} \in \mathcal{Y}(\mathbf{w})} \theta \cdot f(\mathbf{w}, \hat{\mathbf{y}})$, plus
 3662 an additional term for the cost. Essentially, we want to train against predictions that are
 3663 strong and wrong: they should score highly according to the model, yet incur a large loss
 3664 with respect to the ground truth. Training adjusts the weights to reduce the score of these
 3665 predictions.

3666 For cost-augmented decoding to be tractable, the cost function must decompose into
 3667 local parts, just as the feature function $f(\cdot)$ does. The Hamming cost, defined above,
 3668 obeys this property. To perform cost-augmented decoding using the Hamming cost, we
 3669 need only to add features $f_m(y_m) = \delta(y_m \neq y_m^{(i)})$, and assign a constant weight of 1 to
 3670 these features. Decoding can then be performed using the Viterbi algorithm.⁸

As with large-margin classifiers, it is possible to formulate the learning problem in an unconstrained form, by combining a regularization term on the weights and a Lagrangian for the constraints:

$$\min_{\theta} \frac{1}{2} \|\theta\|_2^2 - C \left(\sum_i \theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) - \max_{\mathbf{y} \in \mathcal{Y}(\mathbf{w}^{(i)})} [\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}) + c(\mathbf{y}^{(i)}, \mathbf{y})] \right), \quad [7.73]$$

3671 In this formulation, C is a parameter that controls the tradeoff between the regularization
 3672 term and the margin constraints. A number of optimization algorithms have been
 3673 proposed for structured support vector machines, some of which are discussed in § 2.3.2.
 3674 An empirical comparison by Kummerfeld et al. (2015) shows that stochastic subgradient
 3675 descent — which is essentially a cost-augmented version of the structured perceptron —
 3676 is highly competitive.

3677 7.5.3 Conditional random fields

3678 The **conditional random field** (CRF; Lafferty et al., 2001) is a conditional probabilistic
 3679 model for sequence labeling; just as structured perceptron is built on the perceptron clas-
 3680 sifier, conditional random fields are built on the logistic regression classifier.⁹ The basic

⁸Are there cost functions that do not decompose into local parts? Suppose we want to assign a constant loss c to any prediction $\hat{\mathbf{y}}$ in which k or more predicted tags are incorrect, and zero loss otherwise. This loss function is combinatorial over the predictions, and thus we cannot decompose it into parts.

⁹The name “Conditional Random Field” is derived from **Markov random fields**, a general class of models in which the probability of a configuration of variables is proportional to a product of scores across pairs (or

3681 probability model is,

$$p(\mathbf{y} \mid \mathbf{w}) = \frac{\exp(\Psi(\mathbf{w}, \mathbf{y}))}{\sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp(\Psi(\mathbf{w}, \mathbf{y}'))}. \quad [7.74]$$

3682 This is almost identical to logistic regression, but because the label space is now tag
 3683 sequences, we require efficient algorithms for both **decoding** (searching for the best tag
 3684 sequence given a sequence of words \mathbf{w} and a model θ) and for **normalizing** (summing
 3685 over all tag sequences). These algorithms will be based on the usual locality assumption
 3686 on the scoring function, $\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m)$.

3687 **7.5.3.1 Decoding in CRFs**

Decoding — finding the tag sequence $\hat{\mathbf{y}}$ that maximizes $p(\mathbf{y} \mid \mathbf{w})$ — is a direct application of the Viterbi algorithm. The key observation is that the decoding problem does not depend on the denominator of $p(\mathbf{y} \mid \mathbf{w})$,

$$\begin{aligned} \hat{\mathbf{y}} &= \operatorname{argmax}_{\mathbf{y}} \log p(\mathbf{y} \mid \mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{y}} \Psi(\mathbf{y}, \mathbf{w}) - \log \sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp \Psi(\mathbf{y}', \mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{y}} \Psi(\mathbf{y}, \mathbf{w}) = \operatorname{argmax}_{\mathbf{y}} \sum_{m=1}^{M+1} s(y_m, y_{m-1}). \end{aligned}$$

3688 This is identical to the decoding problem for structured perceptron, so the same Viterbi
 3689 recurrence as defined in Equation 7.22 can be used.

3690 **7.5.3.2 Learning in CRFs**

As with logistic regression, the weights θ are learned by minimizing the regularized negative log-probability,

$$\ell = \frac{\lambda}{2} \|\theta\|^2 - \sum_{i=1}^N \log p(\mathbf{y}^{(i)} \mid \mathbf{w}^{(i)}; \theta) \quad [7.75]$$

$$= \frac{\lambda}{2} \|\theta\|^2 - \sum_{i=1}^N \theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) + \log \sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w}^{(i)})} \exp (\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}')), \quad [7.76]$$

more generally, cliques) of variables in a **factor graph**. In sequence labeling, the pairs of variables include all adjacent tags (y_m, y_{m-1}). The probability is *conditioned* on the words \mathbf{w} , which are always observed, motivating the term “conditional” in the name.

3691 where λ controls the amount of regularization. The final term in Equation 7.76 is a sum
 3692 over all possible labelings. This term is the log of the denominator in Equation 7.74, some-
 3693 times known as the **partition function**.¹⁰ There are $|\mathcal{Y}|^M$ possible labelings of an input of
 3694 size M , so we must again exploit the decomposition of the scoring function to compute
 3695 this sum efficiently.

The sum $\sum_{\mathbf{y} \in \mathcal{Y}^{w(i)}} \exp \Psi(\mathbf{y}, \mathbf{w})$ can be computed efficiently using the **forward recurrence**, which is closely related to the Viterbi recurrence. We first define a set of **forward variables**, $\alpha_m(y_m)$, which is equal to the sum of the scores of all paths leading to tag y_m at position m :

$$\alpha_m(y_m) \triangleq \sum_{\mathbf{y}_{1:m-1}} \exp \sum_{n=1}^m s_n(y_n, y_{n-1}) \quad [7.77]$$

$$= \sum_{\mathbf{y}_{1:m-1}} \prod_{n=1}^m \exp s_n(y_n, y_{n-1}). \quad [7.78]$$

Note the similarity to the definition of the Viterbi variable, $v_m(y_m) = \max_{\mathbf{y}_{1:m-1}} \sum_{n=1}^m s_n(y_n, y_{n-1})$. In the hidden Markov model, the Viterbi recurrence had an alternative interpretation as the max-product algorithm (see Equation 7.53); analogously, the forward recurrence is known as the **sum-product algorithm**, because of the form of [7.78]. The forward variable can also be computed through a recurrence:

$$\alpha_m(y_m) = \sum_{\mathbf{y}_{1:m-1}} \prod_{n=1}^m \exp s_n(y_n, y_{n-1}) \quad [7.79]$$

$$= \sum_{y_{m-1}} (\exp s_m(y_m, y_{m-1})) \sum_{\mathbf{y}_{1:m-2}} \prod_{n=1}^{m-1} \exp s_n(y_n, y_{n-1}) \quad [7.80]$$

$$= \sum_{y_{m-1}} (\exp s_m(y_m, y_{m-1})) \times \alpha_{m-1}(y_{m-1}). \quad [7.81]$$

Using the forward recurrence, it is possible to compute the denominator of the conditional probability,

$$\sum_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} \Psi(\mathbf{w}, \mathbf{y}) = \sum_{\mathbf{y}_{1:M}} s_{M+1}(\blacklozenge, y_M) \prod_{m=1}^M s_m(y_m, y_{m-1}) \quad [7.82]$$

$$= \alpha_{M+1}(\blacklozenge). \quad [7.83]$$

¹⁰The terminology of “potentials” and “partition functions” comes from statistical mechanics (Bishop, 2006).

The conditional log-likelihood can be rewritten,

$$\ell = \frac{\lambda}{2} \|\boldsymbol{\theta}\|^2 - \sum_{i=1}^N \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) + \log \alpha_{M+1}(\blacklozenge). \quad [7.84]$$

- 3696 Probabilistic programming environments, such as `Torch` (Collobert et al., 2011) and `dynet` (Neu-
 3697 big et al., 2017), can compute the gradient of this objective using automatic differentiation.
 3698 The programmer need only implement the forward algorithm as a computation graph.

As in logistic regression, the gradient of the likelihood with respect to the parameters is a difference between observed and expected feature counts:

$$\frac{d\ell}{d\theta_j} = \lambda \theta_j + \sum_{i=1}^N E[f_j(\mathbf{w}^{(i)}, \mathbf{y})] - f_j(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}), \quad [7.85]$$

- 3699 where $f_j(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})$ refers to the count of feature j for token sequence $\mathbf{w}^{(i)}$ and tag se-
 3700 quence $\mathbf{y}^{(i)}$. The expected feature counts are computed “under the hood” when automatic
 3701 differentiation is applied to Equation 7.84 (Eisner, 2016).

- 3702 Before the widespread use of automatic differentiation, it was common to compute
 3703 the feature expectations from marginal tag probabilities $p(y_m | \mathbf{w})$. These marginal prob-
 3704 abilities are sometimes useful on their own, and can be computed using the **forward-**
 3705 **backward algorithm**. This algorithm combines the forward recurrence with an equivalent
 3706 **backward recurrence**, which traverses the input from w_M back to w_1 .

3707 7.5.3.3 *Forward-backward algorithm

Marginal probabilities over tag bigrams can be written as,¹¹

$$\Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}) = \frac{\sum_{\mathbf{y}: Y_m=k, Y_{m-1}=k'} \prod_{n=1}^M \exp s_n(y_n, y_{n-1})}{\sum_{\mathbf{y}'} \prod_{n=1}^M \exp s_n(y'_n, y'_{n-1})}. \quad [7.86]$$

The numerator sums over all tag sequences that include the transition $(Y_{m-1} = k') \rightarrow (Y_m = k)$. Because we are only interested in sequences that include the tag bigram, this sum can be decomposed into three parts: the *prefixes* $\mathbf{y}_{1:m-1}$, terminating in $Y_{m-1} = k'$; the

¹¹Recall the notational convention of upper-case letters for random variables, e.g. Y_m , and lower case letters for specific values, e.g., y_m , so that $Y_m = k$ is interpreted as the event of random variable Y_m taking the value k .

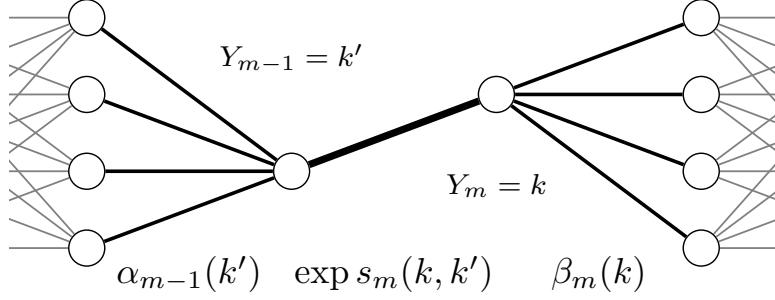


Figure 7.3: A schematic illustration of the computation of the marginal probability $\Pr(Y_{m-1} = k', Y_m = k)$, using the forward score $\alpha_{m-1}(k')$ and the backward score $\beta_m(k)$.

transition $(Y_{m-1} = k') \rightarrow (Y_m = k)$; and the *suffixes* $\mathbf{y}_{m:M}$, beginning with the tag $Y_m = k$:

$$\sum_{\mathbf{y}: Y_m = k, Y_{m-1} = k'} \prod_{n=1}^M \exp s_n(y_n, y_{n-1}) = \sum_{\mathbf{y}_{1:m-1}: Y_{m-1} = k'} \prod_{n=1}^{m-1} \exp s_n(y_n, y_{n-1}) \times \exp s_m(k, k') \times \sum_{\mathbf{y}_{m:M}: Y_m = k} \prod_{n=m+1}^{M+1} \exp s_n(y_n, y_{n-1}). \quad [7.87]$$

The result is product of three terms: a score that sums over all the ways to get to the position $(Y_{m-1} = k')$, a score for the transition from k' to k , and a score that sums over all the ways of finishing the sequence from $(Y_m = k)$. The first term of Equation 7.87 is equal to the **forward variable**, $\alpha_{m-1}(k')$. The third term — the sum over ways to finish the sequence — can also be defined recursively, this time moving over the trellis from right to left, which is known as the **backward recurrence**:

$$\beta_m(k) \triangleq \sum_{\mathbf{y}_{m:M}: Y_m = k} \prod_{n=m}^{M+1} \exp s_n(y_n, y_{n-1}) \quad [7.88]$$

$$= \sum_{k' \in \mathcal{Y}} \exp s_{m+1}(k', k) \sum_{\mathbf{y}_{m+1:M}: Y_m = k'} \prod_{n=m+1}^{M+1} \exp s_n(y_n, y_{n-1}) \quad [7.89]$$

$$= \sum_{k' \in \mathcal{Y}} \exp s_{m+1}(k', k) \times \beta_{m+1}(k'). \quad [7.90]$$

3708 To understand this computation, compare with the forward recurrence in Equation 7.81.

In practice, numerical stability demands that we work in the log domain,

$$\log \alpha_m(k) = \log \sum_{k' \in \mathcal{Y}} \exp (\log s_m(k, k') + \log \alpha_{m-1}(k')) \quad [7.91]$$

$$\log \beta_{m-1}(k) = \log \sum_{k' \in \mathcal{Y}} \exp (\log s_m(k', k) + \log \beta_m(k')). \quad [7.92]$$

The application of the forward and backward probabilities is shown in Figure 7.3. Both the forward and backward recurrences operate on the trellis, which implies a space complexity $\mathcal{O}(MK)$. Because both recurrences require computing a sum over K terms at each node in the trellis, their time complexity is $\mathcal{O}(MK^2)$.

7.6 Neural sequence labeling

In neural network approaches to sequence labeling, we construct a vector representation for each tagging decision, based on the word and its context. Neural networks can perform tagging as a per-token classification decision, or they can be combined with the Viterbi algorithm to tag the entire sequence globally.

7.6.1 Recurrent neural networks

Recurrent neural networks (RNNs) were introduced in chapter 6 as a language modeling technique, in which the context at token m is summarized by a recurrently-updated vector,

$$\mathbf{h}_m = g(\mathbf{x}_m, \mathbf{h}_{m-1}), \quad m = 1, 2, \dots, M,$$

where \mathbf{x}_m is the vector **embedding** of the token w_m and the function g defines the recurrence. The starting condition \mathbf{h}_0 is an additional parameter of the model. The long short-term memory (LSTM) is a more complex recurrence, in which a memory cell is through a series of gates, avoiding repeated application of the non-linearity. Despite these bells and whistles, both models share the basic architecture of recurrent updates across a sequence, and both will be referred to as RNNs here.

A straightforward application of RNNs to sequence labeling is to score each tag y_m as a linear function of \mathbf{h}_m :

$$\psi_m(y) = \beta_y \cdot \mathbf{h}_m \quad [7.93]$$

$$\hat{y}_m = \underset{y}{\operatorname{argmax}} \psi_m(y). \quad [7.94]$$

The score $\psi_m(y)$ can also be converted into a probability distribution using the usual softmax operation,

$$p(y | \mathbf{w}_{1:m}) = \frac{\exp \psi_m(y)}{\sum_{y' \in \mathcal{Y}} \exp \psi_m(y')}. \quad [7.95]$$

3727 Using this transformation, it is possible to train the tagger from the negative log-likelihood
 3728 of the tags, as in a conditional random field. Alternatively, a hinge loss or margin loss
 3729 objective can be constructed from the raw scores $\psi_m(y)$.

The hidden state \mathbf{h}_m accounts for information in the input leading up to position m , but it ignores the subsequent tokens, which may also be relevant to the tag y_m . This can be addressed by adding a second RNN, in which the input is reversed, running the recurrence from w_M to w_1 . This is known as a **bidirectional recurrent neural network** (Graves and Schmidhuber, 2005), and is specified as:

$$\overleftarrow{\mathbf{h}}_m = g(\mathbf{x}_m, \overleftarrow{\mathbf{h}}_{m+1}), \quad m = 1, 2, \dots, M. \quad [7.96]$$

3730 The hidden states of the left-to-right RNN are denoted $\overrightarrow{\mathbf{h}}_m$. The left-to-right and right-to-
 3731 left vectors are concatenated, $\mathbf{h}_m = [\overleftarrow{\mathbf{h}}_m; \overrightarrow{\mathbf{h}}_m]$. The scoring function in Equation 7.93 is
 3732 applied to this concatenated vector.

3733 Bidirectional RNN tagging has several attractive properties. Ideally, the representa-
 3734 tion \mathbf{h}_m summarizes the useful information from the surrounding context, so that it is not
 3735 necessary to design explicit features to capture this information. If the vector \mathbf{h}_m is an ad-
 3736 equate summary of this context, then it may not even be necessary to perform the tagging
 3737 jointly: in general, the gains offered by joint tagging of the entire sequence are diminished
 3738 as the individual tagging model becomes more powerful. Using backpropagation, the
 3739 word vectors \mathbf{x} can be trained “end-to-end”, so that they capture word properties that are
 3740 useful for the tagging task. Alternatively, if limited labeled data is available, we can use
 3741 word embeddings that are “pre-trained” from unlabeled data, using a language modeling
 3742 objective (as in § 6.3) or a related word embedding technique (see chapter 14). It is even
 3743 possible to combine both fine-tuned and pre-trained embeddings in a single model.

3744 **Neural structure prediction** The bidirectional recurrent neural network incorporates in-
 3745 formation from throughout the input, but each tagging decision is made independently.
 3746 In some sequence labeling applications, there are very strong dependencies between tags:
 3747 it may even be impossible for one tag to follow another. In such scenarios, the tagging
 3748 decision must be made jointly across the entire sequence.

3749 Neural sequence labeling can be combined with the Viterbi algorithm by defining the
 3750 local scores as:

$$s_m(y_m, y_{m-1}) = \beta_{y_m} \cdot \mathbf{h}_m + \eta_{y_{m-1}, y_m}, \quad [7.97]$$

3751 where \mathbf{h}_m is the RNN hidden state, β_{y_m} is a vector associated with tag y_m , and η_{y_{m-1}, y_m}
 3752 is a scalar parameter for the tag transition (y_{m-1}, y_m) . These local scores can then be
 3753 incorporated into the Viterbi algorithm for inference, and into the forward algorithm for
 3754 training. This model is shown in Figure 7.4. It can be trained from the conditional log-
 3755 likelihood objective defined in Equation 7.76, backpropagating to the tagging parameters

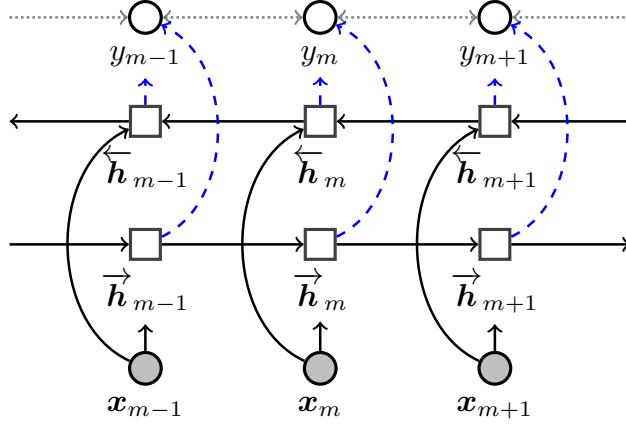


Figure 7.4: Bidirectional LSTM for sequence labeling. The solid lines indicate computation, the dashed lines indicate probabilistic dependency, and the dotted lines indicate the optional additional probabilistic dependencies between labels in the biLSTM-CRF.

3756 β and η , as well as the parameters of the RNN. This model is called the **LSTM-CRF**, due
 3757 to its combination of aspects of the long short-term memory and conditional random field
 3758 models (Huang et al., 2015).

3759 The LSTM-CRF is especially effective on the task of **named entity recognition** (Lample
 3760 et al., 2016), a sequence labeling task that is described in detail in § 8.3. This task has strong
 3761 dependencies between adjacent tags, so structure prediction is especially important.

3762 7.6.2 Character-level models

3763 As in language modeling, rare and unseen words are a challenge: if we encounter a word
 3764 that was not in the training data, then there is no obvious choice for the word embed-
 3765 ding x_m . One solution is to use a generic **unseen word** embedding for all such words.
 3766 However, in many cases, properties of unseen words can be guessed from their spellings.
 3767 For example, *whimsical* does not appear in the Universal Dependencies (UD) English Tree-
 3768 bank, yet the suffix *-al* makes it likely to be adjective; by the same logic, *unflinchingly* is
 3769 likely to be an adverb, and *barnacle* is likely to be a noun.

3770 In feature-based models, these morphological properties were handled by suffix fea-
 3771 tures; in a neural network, they can be incorporated by constructing the embeddings of
 3772 unseen words from their spellings or morphology. One way to do this is to incorporate
 3773 an additional layer of bidirectional RNNs, one for each word in the vocabulary (Ling
 3774 et al., 2015). For each such character-RNN, the inputs are the characters, and the output
 3775 is the concatenation of the final states of the left-facing and right-facing passes, $\phi_w =$

[$\vec{h}_{N_w}^{(w)}; \overleftarrow{h}_0^{(w)}$], where $\vec{h}_{N_w}^{(w)}$ is the final state of the right-facing pass for word w , and N_w is the number of characters in the word. The character RNN model is trained by back-propagation from the tagging objective. On the test data, the trained RNN is applied to out-of-vocabulary words (or all words), yielding inputs to the word-level tagging RNN. Other approaches to compositional word embeddings are described in § 14.7.1.

7.6.3 Convolutional Neural Networks for Sequence Labeling

One disadvantage of recurrent neural networks is that the architecture requires iterating through the sequence of inputs and predictions: each hidden vector h_m must be computed from the previous hidden vector h_{m-1} , before predicting the tag y_m . These iterative computations are difficult to parallelize, and fail to exploit the speedups offered by **graphics processing units (GPUs)** on operations such as matrix multiplication. **Convolutional neural networks** achieve better computational performance by predicting each label y_m from a set of matrix operations on the neighboring word embeddings, $x_{m-k:m+k}$ (Collobert et al., 2011). Because there is no hidden state to update, the predictions for each y_m can be computed in parallel. For more on convolutional neural networks, see § 3.4. Character-based word embeddings can also be computed using convolutional neural networks (Santos and Zadrozny, 2014).

7.7 *Unsupervised sequence labeling

In unsupervised sequence labeling, the goal is to induce a hidden Markov model from a corpus of *unannotated* text ($w^{(1)}, w^{(2)}, \dots, w^{(N)}$), where each $w^{(i)}$ is a sequence of length $M^{(i)}$. This is an example of the general problem of **structure induction**, which is the unsupervised version of structure prediction. The tags that result from unsupervised sequence labeling might be useful for some downstream task, or they might help us to better understand the language’s inherent structure. For part-of-speech tagging, it is common to use a tag dictionary that lists the allowed tags for each word, simplifying the problem (Christodoulopoulos et al., 2010).

Unsupervised learning in hidden Markov models can be performed using the **Baum-Welch algorithm**, which combines the forward-backward algorithm (§ 7.5.3.3) with expectation-maximization (EM; § 5.1.2). In the M-step, the HMM parameters from expected counts:

$$\Pr(W = i \mid Y = k) = \phi_{k,i} = \frac{E[\text{count}(W = i, Y = k)]}{E[\text{count}(Y = k)]}$$

$$\Pr(Y_m = k \mid Y_{m-1} = k') = \lambda_{k',k} = \frac{E[\text{count}(Y_m = k, Y_{m-1} = k')]}{E[\text{count}(Y_{m-1} = k')]} \quad \text{[Equation 7.10]}$$

3802 The expected counts are computed in the E-step, using the forward and backward
 3803 recurrences. The local scores follow the usual definition for hidden Markov models,

$$s_m(k, k') = \log p_E(w_m | Y_m = k; \phi) + \log p_T(Y_m = k | Y_{m-1} = k'; \lambda). \quad [7.98]$$

The expected transition counts for a single instance are,

$$E[\text{count}(Y_m = k, Y_{m-1} = k') | \mathbf{w}] = \sum_{m=1}^M \Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}) \quad [7.99]$$

$$= \frac{\sum_{\mathbf{y}: Y_m=k, Y_{m-1}=k'} \prod_{n=1}^M \exp s_n(y_n, y_{n-1})}{\sum_{\mathbf{y}'} \prod_{n=1}^M \exp s_n(y'_n, y'_{n-1})}. \quad [7.100]$$

As described in § 7.5.3.3, these marginal probabilities can be computed from the forward-backward recurrence,

$$\Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}) = \frac{\alpha_{m-1}(k') \times s_m(k, k') \times \beta_m(k)}{\alpha_{M+1}(\blacklozenge)}. \quad [7.101]$$

In a hidden Markov model, each element of the forward-backward computation has a special interpretation:

$$\alpha_{m-1}(k') = p(Y_{m-1} = k', \mathbf{w}_{1:m-1}) \quad [7.102]$$

$$s_m(k, k') = p(Y_m = k, w_m | Y_{m-1} = k') \quad [7.103]$$

$$\beta_m(k) = p(\mathbf{w}_{m+1:M} | Y_m = k). \quad [7.104]$$

Applying the conditional independence assumptions of the hidden Markov model (defined in Algorithm 12), the product is equal to the joint probability of the tag bigram and the entire input,

$$\begin{aligned} \alpha_{m-1}(k') \times s_m(k, k') \times \beta_m(k) &= p(Y_{m-1} = k', \mathbf{w}_{1:m-1}) \\ &\quad \times p(Y_m = k, w_m | Y_{m-1} = k') \\ &\quad \times p(\mathbf{w}_{m+1:M} | Y_m = k) \\ &= p(Y_{m-1} = k', Y_m = k, \mathbf{w}_{1:M}). \end{aligned} \quad [7.105]$$

Dividing by $\alpha_{M+1}(\blacklozenge) = p(\mathbf{w}_{1:M})$ gives the desired probability,

$$\frac{\alpha_{m-1}(k') \times s_m(k, k') \times \beta_m(k)}{\alpha_{M+1}(\blacklozenge)} = \frac{p(Y_{m-1} = k', Y_m = k, \mathbf{w}_{1:M})}{p(\mathbf{w}_{1:M})} \quad [7.106]$$

$$= \Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}_{1:M}). \quad [7.107]$$

3804 The expected emission counts can be computed in a similar manner, using the product
 3805 $\alpha_m(k) \times \beta_m(k)$.

3806 **7.7.1 Linear dynamical systems**

3807 The forward-backward algorithm can be viewed as Bayesian state estimation in a discrete
 3808 state space. In a continuous state space, $\mathbf{y}_m \in \mathbb{R}^K$, the equivalent algorithm is the **Kalman**
 3809 **smoother**. It also computes marginals $p(\mathbf{y}_m | \mathbf{x}_{1:M})$, using a similar two-step algorithm
 3810 of forward and backward passes. Instead of computing a trellis of values at each step, the
 3811 Kalman smoother computes a probability density function $q_{\mathbf{y}_m}(\mathbf{y}_m; \boldsymbol{\mu}_m, \Sigma_m)$, character-
 3812 ized by a mean $\boldsymbol{\mu}_m$ and a covariance Σ_m around the latent state. Connections between the
 3813 Kalman Smoother and the forward-backward algorithm are elucidated by Minka (1999)
 3814 and Murphy (2012).

3815 **7.7.2 Alternative unsupervised learning methods**

As noted in § 5.5, expectation-maximization is just one of many techniques for structure induction. One alternative is to use **Markov Chain Monte Carlo (MCMC)** sampling algorithms, which are briefly described in § 5.5.1. For the specific case of sequence labeling, Gibbs sampling can be applied by iteratively sampling each tag y_m conditioned on all the others (Finkel et al., 2005):

$$p(y_m | \mathbf{y}_{-m}, \mathbf{w}_{1:M}) \propto p(w_m | y_m) p(y_m | \mathbf{y}_{-m}). \quad [7.108]$$

3816 Gibbs Sampling has been applied to unsupervised part-of-speech tagging by Goldwater
 3817 and Griffiths (2007). **Beam sampling** is a more sophisticated sampling algorithm, which
 3818 randomly draws entire sequences $\mathbf{y}_{1:M}$, rather than individual tags y_m ; this algorithm
 3819 was applied to unsupervised part-of-speech tagging by Van Gael et al. (2009). Spectral
 3820 learning (see § 5.5.2) can also be applied to sequence labeling. By factoring matrices of
 3821 co-occurrence counts of word bigrams and trigrams (Song et al., 2010; Hsu et al., 2012), it
 3822 is possible to obtain globally optimal estimates of the transition and emission parameters,
 3823 under mild assumptions.

3824 **7.7.3 Semiring Notation and the Generalized Viterbi Algorithm**

The Viterbi and Forward recurrences can each be performed over probabilities or log probabilities, yielding a total of four closely related recurrences. These four recurrence scan in fact be expressed as a single recurrence in a more general notation, known as **semiring algebra**. Let the symbol \oplus represent generalized addition, and the symbol \otimes represent generalized multiplication.¹² Given these operators, we can denote a general-

¹²In a semiring, the addition and multiplication operators must both obey associativity, and multiplication must distribute across addition; the addition operator must be commutative; there must be additive and multiplicative identities $\bar{0}$ and $\bar{1}$, such that $a \oplus \bar{0} = a$ and $a \otimes \bar{1} = a$; and there must be a multiplicative annihilator $\bar{0}$, such that $a \otimes \bar{0} = \bar{0}$.

ized Viterbi recurrence as,

$$v_m(k) = \bigoplus_{k' \in \mathcal{Y}} s_m(k, k') \otimes v_{m-1}(k'). \quad [7.109]$$

3825 Each recurrence that we have seen so far is a special case of this generalized Viterbi
3826 recurrence:

- 3827 • In the max-product Viterbi recurrence over probabilities, the \oplus operation corre-
3828 sponds to maximization, and the \otimes operation corresponds to multiplication.
- 3829 • In the forward recurrence over probabilities, the \oplus operation corresponds to addi-
3830 tion, and the \otimes operation corresponds to multiplication.
- 3831 • In the max-product Viterbi recurrence over log-probabilities, the \oplus operation corre-
3832 sponds to maximization, and the \otimes operation corresponds to addition.¹³
- 3833 • In the forward recurrence over log-probabilities, the \oplus operation corresponds to log-
3834 addition, $a \oplus b = \log(e^a + e^b)$. The \otimes operation corresponds to addition.

3835 The mathematical abstraction offered by semiring notation can be applied to the soft-
3836 ware implementations of these algorithms, yielding concise and modular implemen-
3837 tations. The OPENFST library (Allauzen et al., 2007) is an example of a software package in
3838 which the algorithms are parametrized by the choice of semiring.

3839 Exercises

- 3840 1. Consider the garden path sentence, *The old man the boat*. Given word-tag and tag-tag
3841 features, what inequality in the weights must hold for the correct tag sequence to
3842 outscore the garden path tag sequence for this example?
- 3843 2. Sketch out an algorithm for a variant of Viterbi that returns the top- n label se-
3844 quences. What is the time and space complexity of this algorithm?
- 3845 3. Show how to compute the marginal probability $\Pr(y_{m-2} = k, y_m = k' \mid \mathbf{w}_{1:M})$, in
3846 terms of the forwards and backward variables, and the potentials $s_n(y_n, y_{n-1})$.
- 3847 4. Suppose you receive a stream of text, where some of tokens have been replaced at
3848 random with *NOISE*. For example:
 - 3849 • Source: *I try all things, I achieve what I can*
 - 3850 • Message received: *I try NOISE NOISE, I NOISE what I NOISE*

¹³This is sometimes called the **tropical semiring**, in honor of the Brazilian mathematician Imre Simon.

3851 Assume you have access to a pre-trained bigram language model, which gives prob-
3852 abilities $p(w_m \mid w_{m-1})$. These probabilities can be assumed to be non-zero for all
3853 bigrams.

- 3854 a) Show how to use the Viterbi algorithm to try to recover the source by maxi-
3855 mizing the bigram language model log-probability. Specifically, set the scores
3856 $s_m(y_m, y_{m-1})$ so that the Viterbi algorithm selects a sequence of words that
3857 maximizes the bigram language model log-probability, *while leaving the non-*
3858 *noise tokens intact*. Your solution should not modify the logic of the Viterbi
3859 algorithm, it should only set the scores $s_m(y_m, y_{m-1})$.
- 3860 b) An alternative solution is to iterate through the text from $m \in \{1, 2, \dots, M\}$,
3861 replacing each noise token with the word that maximizes $P(w_m \mid w_{m-1})$ ac-
3862 cording to the bigram language model. Given an upper bound on the expected
3863 fraction of tokens for which the two approaches will disagree.
- 3864 5. Consider an RNN tagging model with a tanh activation function on the hidden
3865 layer, and a hinge loss on the output. (The problem also works for the margin loss
3866 and negative log-likelihood.) Suppose you initialize all parameters to zero: this
3867 includes the word embeddings that make up \mathbf{x} , the transition matrix Θ , the out-
3868 put weights β , and the initial hidden state \mathbf{h}_0 . Prove that for any data and for any
3869 gradient-based learning algorithm, all parameters will be stuck at zero.
3870 Extra credit: would a sigmoid activation function avoid this problem?

3871 Chapter 8

3872 Applications of sequence labeling

3873 Sequence labeling has applications throughout natural language processing. This chapter
3874 focuses on part-of-speech tagging, morpho-syntactic attribute tagging, named entity
3875 recognition, and tokenization. It also touches briefly on two applications to interactive
3876 settings: dialogue act recognition and the detection of code-switching points between
3877 languages.

3878 8.1 Part-of-speech tagging

3879 The **syntax** of a language is the set of principles under which sequences of words are
3880 judged to be grammatically acceptable by fluent speakers. One of the most basic syntactic
3881 concepts is the **part-of-speech** (POS), which refers to the syntactic role of each word in a
3882 sentence. This concept was used informally in the previous chapter, and you may have
3883 some intuitions from your own study of English. For example, in the sentence *We like*
3884 *vegetarian sandwiches*, you may already know that *we* and *sandwiches* are nouns, *like* is a
3885 verb, and *vegetarian* is an adjective. These labels depend on the context in which the word
3886 appears: in *she eats like a vegetarian*, the word *like* is a preposition, and the word *vegetarian*
3887 is a noun.

3888 Parts-of-speech can help to disentangle or explain various linguistic problems. Recall
3889 Chomsky's proposed distinction in chapter 6:

- 3890 (8.1) Colorless green ideas sleep furiously.
- 3891 (8.2) *Ideas colorless furiously green sleep.

3892 One difference between these two examples is that the first contains part-of-speech transitions
3893 that are typical in English: adjective to adjective, adjective to noun, noun to verb, and verb
3894 to adverb. The second example contains transitions that are unusual: noun to adjective
3895 and adjective to verb. The ambiguity in a headline like,

3896 (8.3) Teacher Strikes Idle Children

3897 can also be explained in terms of parts of speech: in the interpretation that was likely
 3898 intended, *strikes* is a noun and *idle* is a verb; in the alternative explanation, *strikes* is a verb
 3899 and *idle* is an adjective.

3900 Part-of-speech tagging is often taken as a early step in a natural language processing
 3901 pipeline. Indeed, parts-of-speech provide features that can be useful for many of the
 3902 tasks that we will encounter later, such as parsing (chapter 10), coreference resolution
 3903 (chapter 15), and relation extraction (chapter 17).

3904 **8.1.1 Parts-of-Speech**

3905 The **Universal Dependencies** project (UD) is an effort to create syntactically-annotated
 3906 corpora across many languages, using a single annotation standard (Nivre et al., 2016). As
 3907 part of this effort, they have designed a part-of-speech **tagset**, which is meant to capture
 3908 word classes across as many languages as possible.¹ This section describes that inventory,
 3909 giving rough definitions for each of tags, along with supporting examples.

3910 Part-of-speech tags are **morphosyntactic**, rather than **semantic**, categories. This means
 3911 that they describe words in terms of how they pattern together and how they are inter-
 3912 nally constructed (e.g., what suffixes and prefixes they include). For example, you may
 3913 think of a noun as referring to objects or concepts, and verbs as referring to actions or
 3914 events. But events can also be nouns:

3915 (8.4) ... the **howling** of the **shrieking** storm.

3916 Here *howling* and *shrieking* are events, but grammatically they act as a noun and adjective
 3917 respectively.

3918 **8.1.1.1 The Universal Dependency part-of-speech tagset**

3919 The UD tagset is broken up into three groups: open class tags, closed class tags, and
 3920 “others.”

3921 **Open class tags** Nearly all languages contain nouns, verbs, adjectives, and adverbs.²
 3922 These are all **open word classes**, because new words can easily be added to them. The
 3923 UD tagset includes two other tags that are open classes: proper nouns and interjections.

3924 • **Nouns** (UD tag: NOUN) tend to describe entities and concepts, e.g.,

¹The UD tagset builds on earlier work from Petrov et al. (2012), in which a set of twelve universal tags was identified by creating mappings from tagsets for individual languages.

²One prominent exception is Korean, which some linguists argue does not have adjectives Kim (2002).

3925 (8.5) **Toes** are scarce among veteran **blubber men**.

3926 In English, nouns tend to follow determiners and adjectives, and can play the subject
 3927 role in the sentence. They can be marked for the plural number by an -s suffix.

- 3928 • **Proper nouns** (PROPN) are tokens in names, which uniquely specify a given entity,

3929 (8.6) “**Moby Dick?**” shouted **Ahab**.

- 3930 • **Verbs** (VERB), according to the UD guidelines, “typically signal events and ac-
 3931 tions.” But they are also defined grammatically: they “can constitute a minimal
 3932 predicate in a clause, and govern the number and types of other constituents which
 3933 may occur in a clause.”³

3934 (8.7) “**Moby Dick?**” shouted Ahab.

3935 (8.8) Shall we **keep chasing** this murderous fish?

3936 English verbs tend to come in between the subject and some number of direct ob-
 3937 jects, depending on the verb. They can be marked for **tense** and **aspect** using suffixes
 3938 such as *-ed* and *-ing*. (These suffixes are an example of **inflectional morphology**,
 3939 which is discussed in more detail in § 9.1.4.)

- 3940 • **Adjectives** (ADJ) describe properties of entities,

3941 (8.9) Shall we keep chasing this **murderous** fish?

3942 (8.10) Toes are **scarce** among **veteran** blubber men.

3943 In the second example, *scarce* is a predicative adjective, linked to the subject by the
 3944 **copula verb** *are*. This means that In contrast, *murderous* and *veteran* are attribute
 3945 adjectives, modifying the noun phrase in which they are embedded.

- 3946 • **Adverbs** (ADV) describe properties of events, and may also modify adjectives or
 3947 other adverbs:

3948 (8.11) It is not down on any map; true places **never** are.

3949 (8.12) ... **treacherously** hidden beneath the loveliest tints of azure

3950 (8.13) Not drowned **entirely**, though.

- 3951 • **Interjections** (INTJ) are used in exclamations, e.g.,

3952 (8.14) **Aye aye!** it was that accursed white whale that razed me.

³<http://universaldependencies.org/u/pos/VERB.html>

3953 **Closed class tags** Closed word classes rarely receive new members. They are sometimes
 3954 referred to as **function words** — as opposed to **content words** — as they have little lexical
 3955 meaning of their own, but rather, help to organize the components of the sentence.

- 3956 • **Adpositions** (ADP) describe the relationship between a complement (usually a noun
 3957 phrase) and another unit in the sentence, typically a noun or verb phrase.

- 3958 (8.15) Toes are scarce **among** veteran blubber men.
 3959 (8.16) It is not **down on** any map.
 3960 (8.17) Give not thyself **up** then.

3961 As the examples show, English generally uses prepositions, which are adpositions
 3962 that appear before their complement. (An exception is *ago*, as in, *we met three days*
 3963 *ago*). Postpositions are used in other languages, such as Japanese and Turkish.

- 3964 • **Auxiliary verbs** (AUX) are a closed class of verbs that add information such as
 3965 tense, aspect, person, and number.

- 3966 (8.18) **Shall** we keep chasing this murderous fish?
 3967 (8.19) What the white whale was to Ahab, **has been** hinted.
 3968 (8.20) Ahab **must** use tools.
 3969 (8.21) Meditation and water **are** wedded forever.
 3970 (8.22) Toes **are** scarce among veteran blubber men.

3971 The final example is a copula verb, which is also tagged as an auxiliary in the UD
 3972 corpus.

- 3973 • **Coordinating conjunctions** (CCONJ) express relationships between two words or
 3974 phrases, which play a parallel role:

- 3975 (8.23) Meditation **and** water are wedded forever.

- 3976 • **Subordinating conjunctions** (SCONJ) link two elements, making one syntactically
 3977 subordinate to the other:

- 3978 (8.24) There is wisdom **that** is woe.

- 3979 • **Pronouns** (PRON) are words that substitute for nouns or noun phrases.

- 3980 (8.25) Be **it what it will**, I'll go to **it** laughing.
 3981 (8.26) I try all things, **I** achieve **what I can**.

3982 The example includes the personal pronouns *I* and *it*, as well as the relative pronoun
 3983 *what*. Other pronouns include *myself*, *somebody*, and *nothing*.

- 3984 • **Determiners** (DET) provide additional information about the nouns or noun phrases
 3985 that they modify:

3986 (8.27) What **the** white whale was to Ahab, has been hinted.

3987 (8.28) It is not down on **any** map.

3988 (8.29) I try **all** things ...

3989 (8.30) Shall we keep chasing **this** murderous fish?

3990 Determiners include articles (*the*), possessive determiners (*their*), demonstratives
 3991 (*this murderous fish*), and quantifiers (*any map*).

- 3992 • **Numerals** (NUM) are an infinite but closed class, which includes integers, fractions,
 3993 and decimals, regardless of whether spelled out or written in numerical form.

3994 (8.31) How then can this **one** small heart beat.

3995 (8.32) I am going to put him down for the **three hundredth**.

- 3996 • **Particles** (PART) are a catch-all of function words that combine with other words or
 3997 phrases, but do not meet the conditions of the other tags. In English, this includes
 3998 the infinitival *to*, the possessive marker, and negation.

3999 (8.33) Better **to** sleep with a sober cannibal than a drunk Christian.

4000 (8.34) So man's insanity is heaven's sense

4001 (8.35) It is **not** down on any map

4002 As the second example shows, the possessive marker is not considered part of the
 4003 same token as the word that it modifies, so that *man's* is split into two tokens. (Tok-
 4004 enization is described in more detail in § 8.4.) A non-English example of a particle
 4005 is the Japanese question marker *ka*, as in,⁴

4006 (8.36) *Sensei desu ka*

Teacher are ?

4007 Is she a teacher?

⁴In this notation, the first line is the transliterated Japanese text, the second line is a token-to-token **gloss**, and the third line is the translation.

4008 **Other** The remaining UD tags include punctuation (PUN) and symbols (SYM). Punc-
 4009 tuation is purely structural — e.g., commas, periods, colons — while symbols can carry
 4010 content of their own. Examples of symbols include dollar and percentage symbols, math-
 4011 ematical operators, emoticons, emojis, and internet addresses. A final catch-all tag is X,
 4012 which is used for words that cannot be assigned another part-of-speech category. The X
 4013 tag is also used in cases of **code switching** (between languages), described in § 8.5.

4014 **8.1.1.2 Other tagsets**

4015 Prior to the Universal Dependency treebank, part-of-speech tagging was performed us-
 4016 ing language-specific tagsets. The dominant tagset for English was designed as part of
 4017 the **Penn Treebank** (PTB), and it includes 45 tags — more than three times as many as
 4018 the UD tagset. This granularity is reflected in distinctions between singular and plural
 4019 nouns, verb tenses and aspects, possessive and non-possessive pronouns, comparative
 4020 and superlative adjectives and adverbs (e.g., *faster, fastest*), and so on. The Brown corpus
 4021 includes a tagset that is even more detailed, with 87 tags Francis (1964), including special
 4022 tags for individual auxiliary verbs such as *be, do, and have*.

4023 Different languages make different distinctions, and so the PTB and Brown tagsets are
 4024 not appropriate for a language such as Chinese, which does not mark the verb tense (Xia,
 4025 2000); nor for Spanish, which marks every combination of person and number in the
 4026 verb ending; nor for German, which marks the case of each noun phrase. Each of these
 4027 languages requires more detail than English in some areas of the tagset, and less in other
 4028 areas. The strategy of the Universal Dependencies corpus is to design a coarse-grained
 4029 tagset to be used across all languages, and then to additionally annotate language-specific
 4030 **morphosyntactic attributes**, such as number, tense, and case. The attribute tagging task
 4031 is described in more detail in § 8.2.

4032 Social media such as Twitter have been shown to require tagsets of their own (Gimpel
 4033 et al., 2011). Such corpora contain some tokens that are not equivalent to anything en-
 4034 countered in a typical written corpus: e.g., emoticons, URLs, and hashtags. Social media
 4035 also includes dialectal words like *gonna* ('going to', e.g. *We gonna be fine*) and *Ima* ('I'm
 4036 going to', e.g., *Ima tell you one more time*), which can be analyzed either as non-standard
 4037 orthography (making tokenization impossible), or as lexical items in their own right. In
 4038 either case, it is clear that existing tags like NOUN and VERB cannot handle cases like *Ima*,
 4039 which combine aspects of the noun and verb. Gimpel et al. (2011) therefore propose a new
 4040 set of tags to deal with these cases.

4041 **8.1.2 Accurate part-of-speech tagging**

4042 Part-of-speech tagging is the problem of selecting the correct tag for each word in a sen-
 4043 tence. Success is typically measured by accuracy on an annotated test set, which is simply
 4044 the fraction of tokens that were tagged correctly.

4045 8.1.2.1 Baselines

4046 A simple baseline for part-of-speech tagging is to choose the most common tag for each
4047 word. For example, in the Universal Dependencies treebank, the word *talk* appears 96
4048 times, and 85 of those times it is labeled as a VERB: therefore, this baseline will always
4049 predict VERB for this word. For words that do not appear in the training corpus, the base-
4050 line simply guesses the most common tag overall, which is NOUN. In the Penn Treebank,
4051 this simple baseline obtains accuracy above 92%. A more rigorous evaluation is the accu-
4052 racy on **out-of-vocabulary words**, which are not seen in the training data. Tagging these
4053 words correctly requires attention to the context and the word's internal structure.

4054 8.1.2.2 Contemporary approaches

4055 Conditional random fields and structured perceptron perform at or near the state-of-the-
4056 art for part-of-speech tagging in English. For example, (Collins, 2002) achieved 97.1%
4057 accuracy on the Penn Treebank, using a structured perceptron with the following base
4058 features (originally introduced by Ratnaparkhi (1996)):

- 4059 • current word, w_m
- 4060 • previous words, w_{m-1}, w_{m-2}
- 4061 • next words, w_{m+1}, w_{m+2}
- 4062 • previous tag, y_{m-1}
- 4063 • previous two tags, (y_{m-1}, y_{m-2})
- 4064 • for rare words:
 - 4065 – first k characters, up to $k = 4$
 - 4066 – last k characters, up to $k = 4$
 - 4067 – whether w_m contains a number, uppercase character, or hyphen.

4068 Similar results for the PTB data have been achieved using conditional random fields (CRFs;
4069 Toutanova et al., 2003).

4070 More recent work has demonstrated the power of neural sequence models, such as the
4071 **long short-term memory (LSTM)** (§ 7.6). Plank et al. (2016) apply a CRF and a bidirec-
4072 tional LSTM to twenty-two languages in the UD corpus, achieving an average accuracy
4073 of 94.3% for the CRF, and 96.5% with the bi-LSTM. Their neural model employs three
4074 types of embeddings: fine-tuned word embeddings, which are updated during training;
4075 pre-trained word embeddings, which are never updated, but which help to tag out-of-
4076 vocabulary words; and character-based embeddings. The character-based embeddings
4077 are computed by running an LSTM on the individual characters in each word, thereby
4078 capturing common orthographic patterns such as prefixes, suffixes, and capitalization.
4079 Extensive evaluations show that these additional embeddings are crucial to their model's
4080 success.

word	PTB tag	UD tag	UD attributes
<i>The</i>	DT	DET	DEFINITE=DEF PRONTYPE=ART
<i>German</i>	JJ	ADJ	DEGREE=POS
<i>Expressionist</i>	NN	NOUN	NUMBER=SING
<i>movement</i>	NN	NOUN	NUMBER=SING
<i>was</i>	VBD	AUX	MOOD=IND NUMBER=SING PERSON=3 TENSE=PAST VERBFORM=FIN
<i>destroyed</i>	VBN	VERB	TENSE=PAST VERBFORM=PART VOICE=PASS
<i>as</i>	IN	ADP	
<i>a</i>	DT	DET	DEFINITE=IND PRONTYPE=ART
<i>result</i>	NN	NOUN	NUMBER=SING
.	.	PUNCT	

Figure 8.1: UD and PTB part-of-speech tags, and UD morphosyntactic attributes. Example selected from the UD 1.4 English corpus.

4081 8.2 Morphosyntactic Attributes

4082 There is considerably more to say about a word than whether it is a noun or a verb: in En-
 4083 glish, verbs are distinguish by features such tense and aspect, nouns by number, adjectives
 4084 by degree, and so on. These features are language-specific: other languages distinguish
 4085 other features, such as **case** (the role of the noun with respect to the action of the sen-
 4086 tence, which is marked in languages such as Latin and German⁵) and **evidentiality** (the
 4087 source of information for the speaker’s statement, which is marked in languages such as
 4088 Turkish). In the UD corpora, these attributes are annotated as feature-value pairs for each
 4089 token.⁶

4090 An example is shown in Figure 8.1. The determiner *the* is marked with two attributes:
 4091 **PRONTYPE=ART**, which indicates that it is an **article** (as opposed to another type of deter-

⁵Case is marked in English for some personal pronouns, e.g., *She saw her, They saw them*.

⁶The annotation and tagging of morphosyntactic attributes can be traced back to earlier work on Turkish (Oflazer and Kuruöz, 1994) and Czech (Hajič and Hladká, 1998). MULTEXT-East was an early multilingual corpus to include morphosyntactic attributes (Dimitrova et al., 1998).

miner or pronominal modifier), and DEFINITE=DEF, which indicates that it is a **definite article** (referring to a specific, known entity). The verbs are each marked with several attributes. The auxiliary verb *was* is third-person, singular, past tense, finite (conjugated), and indicative (describing an event that has happened or is currently happenings); the main verb *destroyed* is in participle form (so there is no additional person and number information), past tense, and passive voice. Some, but not all, of these distinctions are reflected in the PTB tags VBD (past-tense verb) and VBN (past participle).

While there are thousands of papers on part-of-speech tagging, there is comparatively little work on automatically labeling morphosyntactic attributes. Faruqui et al. (2016) train a support vector machine classification model, using a minimal feature set that includes the word itself, its prefixes and suffixes, and type-level information listing all possible morphosyntactic attributes for each word and its neighbors. Mueller et al. (2013) use a conditional random field (CRF), in which the tag space consists of all observed combinations of morphosyntactic attributes (e.g., the tag would be DEF+ART for the word *the* in Figure 8.1). This massive tag space is managed by decomposing the feature space over individual attributes, and pruning paths through the trellis. More recent work has employed bidirectional LSTM sequence models. For example, Pinter et al. (2017) train a bidirectional LSTM sequence model. The input layer and hidden vectors in the LSTM are shared across attributes, but each attribute has its own output layer, culminating in a softmax over all attribute values, e.g. $y_t^{\text{NUMBER}} \in \{\text{SING}, \text{PLURAL}, \dots\}$. They find that character-level information is crucial, especially when the amount of labeled data is limited.

Evaluation is performed by first computing recall and precision for each attribute. These scores can then be averaged at either the type or token level to obtain micro- or macro-*F*-MEASURE. Pinter et al. (2017) evaluate on 23 languages in the UD treebank, reporting a median micro-*F*-MEASURE of 0.95. Performance is strongly correlated with the size of the labeled dataset for each language, with a few outliers: for example, Chinese is particularly difficult, because although the dataset is relatively large (10^5 tokens in the UD 1.4 corpus), only 6% of tokens have any attributes, offering few useful labeled instances.

8.3 Named Entity Recognition

A classical problem in information extraction is to recognize and extract mentions of **named entities** in text. In news documents, the core entity types are people, locations, and organizations; more recently, the task has been extended to include amounts of money, percentages, dates, and times. In item 8.37 (Figure 8.2), the named entities include: *The U.S. Army*, an organization; *Atlanta*, a location; and *May 14, 1864*, a date. Named entity recognition is also a key task in **biomedical natural language processing**, with entity types including proteins, DNA, RNA, and cell lines (e.g., Collier et al., 2000; Ohta et al., 2002). Figure 8.2 shows an example from the GENIA corpus of biomedical research ab-

- (8.37) *The U.S. Army captured Atlanta on May 14, 1864*
 B-ORG I-ORG I-ORG O B-LOC O B-DATE I-DATE I-DATE I-DATE
 (8.38) *Number of glucocorticoid receptors in lymphocytes and ...*
 O O B-PROTEIN I-PROTEIN O B-CELLTYPE O ...

Figure 8.2: BIO notation for named entity recognition. Example (8.38) is drawn from the GENIA corpus of biomedical documents (Ohta et al., 2002).

4130 stracts.

4131 A standard approach to tagging named entity spans is to use discriminative sequence
 4132 labeling methods such as conditional random fields. However, the named entity recogni-
 4133 tion (NER) task would seem to be fundamentally different from sequence labeling tasks
 4134 like part-of-speech tagging: rather than tagging each token, the goal is to recover *spans*
 4135 of tokens, such as *The United States Army*.

4136 This is accomplished by the **BIO notation**, shown in Figure 8.2. Each token at the
 4137 beginning of a name span is labeled with a B- prefix; each token within a name span is la-
 4138 beled with an I- prefix. These prefixes are followed by a tag for the entity type, e.g. B-LOC
 4139 for the beginning of a location, and I-PROTEIN for the inside of a protein name. Tokens
 4140 that are not parts of name spans are labeled as O. From this representation, the entity
 4141 name spans can be recovered unambiguously. This tagging scheme is also advantageous
 4142 for learning: tokens at the beginning of name spans may have different properties than
 4143 tokens within the name, and the learner can exploit this. This insight can be taken even
 4144 further, with special labels for the last tokens of a name span, and for unique tokens in
 4145 name spans, such as *Atlanta* in the example in Figure 8.2. This is called BILOU notation,
 4146 and it can yield improvements in supervised named entity recognition (Ratinov and Roth,
 4147 2009).

Feature-based sequence labeling Named entity recognition was one of the first applications of conditional random fields (McCallum and Li, 2003). The use of Viterbi decoding restricts the feature function $f(\mathbf{w}, \mathbf{y})$ to be a sum of local features, $\sum_m f(\mathbf{w}, y_m, y_{m-1}, m)$, so that each feature can consider only local adjacent tags. Typical features include tag transitions, word features for w_m and its neighbors, character-level features for prefixes and suffixes, and “word shape” features for capitalization and other orthographic properties. As an example, base features for the word *Army* in the example in (8.37) include:

(CURR-WORD:*Army*, PREV-WORD:*U.S.*, NEXT-WORD:*captured*, PREFIX-1:*A-*,
 PREFIX-2:*Ar-*, SUFFIX-1:*-y*, SUFFIX-2:*-my*, SHAPE:*Xxxx*)

4148 Another source of features is to use **gazetteers**: lists of known entity names. For example,
 4149 the U.S. Social Security Administration provides a list of tens of thousands of given names

- (1) 日文 章魚 怎麼 說?
 Japanese octopus how say
 How to say octopus in Japanese?
- (2) 日 文章 魚 怎麼 說?
 Japan essay fish how say

Figure 8.3: An example of tokenization ambiguity in Chinese (Sproat et al., 1996)

4150 — more than could be observed in any annotated corpus. Tokens or spans that match an
 4151 entry in a gazetteer can receive special features; this provides a way to incorporate hand-
 4152 crafted resources such as name lists in a learning-driven framework.

4153 **Neural sequence labeling for NER** Current research has emphasized neural sequence
 4154 labeling, using similar LSTM models to those employed in part-of-speech tagging (Ham-
 4155 merton, 2003; Huang et al., 2015; Lample et al., 2016). The bidirectional LSTM-CRF (Fig-
 4156 ure 7.4 in § 7.6) does particularly well on this task, due to its ability to model tag-to-tag
 4157 dependencies. However, Strubell et al. (2017) show that **convolutional neural networks**
 4158 can be equally accurate, with significant improvement in speed due to the efficiency of
 4159 implementing ConvNets on **graphics processing units (GPUs)**. The key innovation in
 4160 this work was the use of **dilated convolution**, which is described in more detail in § 3.4.

4161 8.4 Tokenization

4162 A basic problem for text analysis, first discussed in § 4.3.1, is to break the text into a se-
 4163 quence of discrete tokens. For alphabetic languages such as English, deterministic scripts
 4164 suffice to achieve accurate tokenization. However, in logographic writing systems such
 4165 as Chinese script, words are typically composed of a small number of characters, with-
 4166 out intervening whitespace. The tokenization must be determined by the reader, with
 4167 the potential for occasional ambiguity, as shown in Figure 8.3. One approach is to match
 4168 character sequences against a known dictionary (e.g., Sproat et al., 1996), using additional
 4169 statistical information about word frequency. However, no dictionary is completely com-
 4170 prehensive, and dictionary-based approaches can struggle with such out-of-vocabulary
 4171 words.

4172 Chinese tokenization has therefore been approached as a supervised sequence label-
 4173 ing problem. Xue et al. (2003) train a logistic regression classifier to make independent
 4174 segmentation decisions while moving a sliding window across the document. A set of
 4175 rules is then used to convert these individual classification decisions into an overall tok-
 4176 enization of the input. However, these individual decisions may be globally suboptimal,
 4177 motivating a structure prediction approach. Peng et al. (2004) train a conditional random

field to predict labels of START or NONSTART on each character. More recent work has employed neural network architectures. For example, Chen et al. (2015) use an LSTM-CRF architecture, as described in § 7.6: they construct a trellis, in which each tag is scored according to the hidden state of an LSTM, and tag-tag transitions are scored according to learned transition weights. The best-scoring segmentation is then computed by the Viterbi algorithm.

4184 8.5 Code switching

4185 Multilingual speakers and writers do not restrict themselves to a single language. **Code**
4186 **switching** is the phenomenon of switching between languages in speech and text (Auer,
4187 2013; Poplack, 1980). Written code switching has become more common in online social
4188 media, as in the following extract from Justin Trudeau's website:⁷

- 4189 (8.39) *Although everything written on this site est disponible en anglais
is available in English
and in French, my personal videos seront bilingues
will be bilingual*

4191 Accurately analyzing such texts requires first determining which languages are being
4192 used. Furthermore, quantitative analysis of code switching can provide insights on the
4193 languages themselves and their relative social positions.

Code switching can be viewed as a sequence labeling problem, where the goal is to label each token as a candidate switch point. In the example above, the words *est*, *and*, and *seront* would be labeled as switch points. Solorio and Liu (2008) detect English-Spanish switch points using a supervised classifier, with features that include the word, its part-of-speech in each language (according to a supervised part-of-speech tagger), and the probabilities of the word and part-of-speech in each language. Nguyen and Dogruöz (2013) apply a conditional random field to the problem of detecting code switching between Turkish and Dutch.

Code switching is a special case of the more general problem of word level language identification, which Barman et al. (2014) address in the context of trilingual code switching between Bengali, English, and Hindi. They further observe an even more challenging phenomenon: intra-word code switching, such as the use of English suffixes with Bengali roots. They therefore mark each token as either (1) belonging to one of the three languages; (2) a mix of multiple languages; (3) “universal” (e.g., symbols, numbers, emoticons); or (4) undefined.

⁷As quoted in <http://blogues.lapresse.ca/lagace/2008/09/08/justin-trudeau-really-parfait-bilingue/>, accessed August 21, 2017.

Speaker	Dialogue Act	Utterance
A	YES-NO-QUESTION	<i>So do you go college right now?</i>
A	ABANDONED	<i>Are yo-</i>
B	YES-ANSWER	<i>Yeah,</i>
B	STATEMENT	<i>It's my last year [laughter].</i>
A	DECLARATIVE-QUESTION	<i>You're a, so you're a senior now.</i>
B	YES-ANSWER	<i>Yeah,</i>
B	STATEMENT	<i>I'm working on my projects trying to graduate [laughter]</i>
A	APPRECIATION	<i>Oh, good for you.</i>
B	BACKCHANNEL	<i>Yeah.</i>

Figure 8.4: An example of dialogue act labeling (Stolcke et al., 2000)

4209 8.6 Dialogue acts

4210 The sequence labeling problems that we have discussed so far have been over sequences
 4211 of word tokens or characters (in the case of tokenization). However, sequence labeling
 4212 can also be performed over higher-level units, such as **utterances**. **Dialogue acts** are la-
 4213 bels over utterances in a dialogue, corresponding roughly to the speaker’s intention —
 4214 the utterance’s **illocutionary force** (Austin, 1962). For example, an utterance may state a
 4215 proposition (*it is not down on any map*), pose a question (*shall we keep chasing this murderous*
 4216 *fish?*), or provide a response (*aye aye!*). Stolcke et al. (2000) describe how a set of 42 dia-
 4217 logue acts were annotated for the 1,155 conversations in the Switchboard corpus (Godfrey
 4218 et al., 1992).⁸

4219 An example is shown in Figure 8.4. The annotation is performed over UTTERANCES,
 4220 with the possibility of multiple utterances per **conversational turn** (in cases such as inter-
 4221 ruptions, an utterance may split over multiple turns). Some utterances are clauses (e.g., *So*
 4222 *do you go to college right now?*), while others are single words (e.g., *yeah*). Stolcke et al. (2000)
 4223 report that hidden Markov models (HMMs) achieve 96% accuracy on supervised utter-
 4224 ance segmentation. The labels themselves reflect the conversational goals of the speaker:
 4225 the utterance *yeah* functions as an answer in response to the question *you’re a senior now*,
 4226 but in the final line of the excerpt, it is a **backchannel** (demonstrating comprehension).

4227 For task of dialogue act labeling, Stolcke et al. (2000) apply a hidden Markov model.
 4228 The probability $p(w_m | y_m)$ must generate the entire sequence of words in the utterance,
 4229 and it is modeled as a trigram language model (§ 6.1). Stolcke et al. (2000) also account
 4230 for acoustic features, which capture the **prosody** of each utterance — for example, tonal
 4231 and rhythmic properties of speech, which can be used to distinguish dialogue acts such

⁸Dialogue act modeling is not restricted to speech; it is relevant in any interactive conversation. For example, Jeong et al. (2009) annotate a more limited set of **speech acts** in a corpus of emails and online forums.

4232 as questions and answers. These features are handled with an additional emission distri-
4233 bution, $p(a_m | y_m)$, which is modeled with a probabilistic decision tree (Murphy, 2012).
4234 While acoustic features yield small improvements overall, they play an important role in
4235 distinguish questions from statements, and agreements from backchannels.

4236 Recurrent neural architectures for dialogue act labeling have been proposed by Kalch-
4237 brenner and Blunsom (2013) and Ji et al. (2016), with strong empirical results. Both models
4238 are recurrent at the utterance level, so that each complete utterance updates a hidden state.
4239 The recurrent-convolutional network of Kalchbrenner and Blunsom (2013) uses convolu-
4240 tion to obtain a representation of each individual utterance, while Ji et al. (2016) use a
4241 second level of recurrence, over individual words. This enables their method to also func-
4242 tion as a language model, giving probabilities over sequences of words in a document.

4243 **Exercises**

- 4244 1. [todo: exercises tk]

4245

Chapter 9

4246

Formal language theory

4247 We have now seen methods for learning to label individual words, vectors of word counts,
4248 and sequences of words; we will soon proceed to more complex structural transfor-
4249 mations. Most of these techniques could apply to counts or sequences from any discrete vo-
4250 cabulary; there is nothing fundamentally linguistic about, say, a hidden Markov model.
4251 This raises a basic question that this text has not yet considered: what is a language?

4252 This chapter will take the perspective of **formal language theory**, in which a language
4253 is defined as a set of **strings**, each of which is a sequence of elements from a finite alphabet.
4254 For interesting languages, there are an infinite number of strings that are in the language,
4255 and an infinite number of strings that are not. For example:

- 4256 • the set of all even-length sequences from the alphabet $\{a, b\}$, e.g., $\{\emptyset, aa, ab, ba, bb, aaaa, aaab, \dots\}$;
- 4257 • the set of all sequences from the alphabet $\{a, b\}$ that contain *aaa* as a substring, e.g.,
4258 $\{aaa, aaaa, baaa, aaab, \dots\}$;
- 4259 • the set of all sequences of English words (drawn from a finite dictionary) that con-
4260 tain at least one verb (a finite subset of the dictionary);
- 4261 • the `python` programming language.

4262 Formal language theory defines classes of languages and their computational prop-
4263 erties. Of particular interest is the computational complexity of solving the **membership**
4264 **problem** — determining whether a string is in a language. The chapter will focus on
4265 three classes of formal languages: regular, context-free, and “mildly” context-sensitive
4266 languages.

4267 A key insight of 20th century linguistics is that formal language theory can be usefully
4268 applied to natural languages such as English, by designing formal languages that cap-
4269 ture as many properties of the natural language as possible. For many such formalisms, a
4270 useful linguistic analysis comes as a byproduct of solving the membership problem. The

4271 membership problem can be generalized to the problems of *scoring* strings for their ac-
 4272 ceptability (as in language modeling), and of **transducing** one string into another (as in
 4273 translation).

4274 9.1 Regular languages

4275 Sooner or later, most computer scientists will write a **regular expression**. If you have,
 4276 then you have defined a **regular language**, which is any language that can be defined by
 4277 a regular expression. Formally, a regular expression can include the following elements:

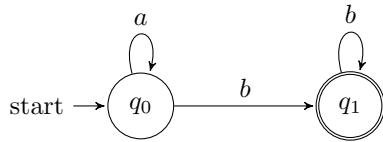
- 4278 • A **literal character** drawn from some finite alphabet Σ .
- 4279 • The **empty string** ϵ .
- 4280 • The concatenation of two regular expressions RS , where R and S are both regular
 4281 expressions. The resulting expression accepts any string that can be decomposed
 4282 $x = yz$, where y is accepted by R and z is accepted by S .
- 4283 • The alternation $R \mid S$, where R and S are both regular expressions. The resulting
 4284 expression accepts a string x if it is accepted by R or it is accepted by S .
- 4285 • The **Kleene star** R^* , which accepts any string x that can be decomposed into a se-
 4286 quence of strings which are all accepted by R .
- 4287 • Parenthesization ((R)), which is used to limit the scope of the concatenation, alterna-
 4288 tion, and Kleene star operators.

4289 Here are some example regular expressions:

- 4290 • The set of all even length strings on the alphabet $\{a, b\}$: $((aa)|(ab)|(ba)|(bb))^*$
- 4291 • The set of all sequences of the alphabet $\{a, b\}$ that contain aaa as a substring: $(a|b)^*aaa(a|b)^*$
- 4292 • The set of all sequences of English words that contain at least one verb: W^*VW^* ,
 4293 where W is an alternation between all words in the dictionary, and V is an alterna-
 4294 tion between all verbs ($V \subseteq W$).

4295 This list does not include a regular expression for the Python programming language,
 4296 because this language is not regular — there is no regular expression that can capture its
 4297 syntax. We will discuss why towards the end of this section.

4298 Regular languages are **closed** under union, intersection, and concatenation. This means,
 4299 for example, that if two languages L_1 and L_2 are regular, then so are the languages $L_1 \cup L_2$,
 4300 $L_1 \cap L_2$, and the language of strings that can be decomposed as $s = tu$, with $s \in L_1$ and
 4301 $t \in L_2$. Regular languages are also closed under negation: if L is regular, then so is the
 4302 language $\bar{L} = \{s \notin L\}$.

Figure 9.1: State diagram for the finite state acceptor M_1 .

4303 **9.1.1 Finite state acceptors**

4304 A regular expression defines a regular language, but does not give an algorithm for de-
 4305 termining whether a string is in the language that it defines. **Finite state automata** are
 4306 theoretical models of computation on regular languages, which involve transitions be-
 4307 tween a finite number of states. The most basic type of finite state automaton is the **finite**
 4308 **state acceptor (FSA)**, which describes the computation involved in testing if a string is
 4309 a member of a language. Formally, a finite state acceptor is a tuple $M = (Q, \Sigma, q_0, F, \delta)$,
 4310 consisting of:

- 4311 • a finite alphabet Σ of input symbols;
- 4312 • a finite set of states $Q = \{q_0, q_1, \dots, q_n\}$;
- 4313 • a start state $q_0 \in Q$;
- 4314 • a set of final states $F \subseteq Q$;
- 4315 • a transition function $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$. The transition function maps from a
 4316 state and an input symbol (or empty string ϵ) to a *set* of possible resulting states.

4317 A **path** in M is a sequence of transitions, $\pi = t_1, t_2, \dots, t_N$, where each t_i traverses an
 4318 arc in the transition function δ . The finite state acceptor M accepts a string ω if there is
 4319 a **accepting path**, in which the initial transition t_1 begins at the start state q_0 , the final
 4320 transition t_N terminates in a final state in Q , and the entire input ω is consumed.

4321 **9.1.1.1 Example**

Consider the following FSA, M_1 .

$$\Sigma = \{a, b\} \quad [9.1]$$

$$Q = \{q_0, q_1\} \quad [9.2]$$

$$F = \{q_1\} \quad [9.3]$$

$$\delta = \{(q_0, a) \rightarrow q_0, (q_0, b) \rightarrow q_1, (q_1, b) \rightarrow q_1\}. \quad [9.4]$$

4322 This FSA defines a language over an alphabet of two symbols, a and b . The transition
 4323 function δ is written as a set of arcs: $(q_0, a) \rightarrow q_0$ says that if the machine is in state

4324 q_0 and reads symbol a , it stays in q_0 . Figure 9.1 provides a graphical representation of
 4325 M_1 . Because each pair of initial state and symbol has at most one resulting state, M_1 is
 4326 **deterministic**: each string ω induces at most one accepting path. Note that there are no
 4327 transitions for the symbol a in state q_1 ; if a is encountered in q_1 , then the acceptor is stuck,
 4328 and the input string is rejected.

4329 What strings does M_1 accept? The start state is q_0 , and we have to get to q_1 , since this
 4330 is the only final state. Any number of a symbols can be consumed in q_0 , but a b symbol is
 4331 required to transition to q_1 . Once there, any number of b symbols can be consumed, but
 4332 an a symbol cannot. So the regular expression corresponding to the language defined by
 4333 M_1 is a^*bb^* .

4334 9.1.1.2 Computational properties of finite state acceptors

4335 The key computational question for finite state acceptors is: how fast can we determine
 4336 whether a string is accepted? For deterministic FSAs, this computation can be performed
 4337 by Dijkstra's algorithm, with time complexity $\mathcal{O}(V \log V + E)$, where V is the number of
 4338 vertices in the FSA, and E is the number of edges (Cormen et al., 2009). Non-deterministic
 4339 FSAs (NFSAs) can include multiple transitions from a given symbol and state. Any NSFA
 4340 can be converted into a deterministic FSA, but the resulting automaton may have a num-
 4341 ber of states that is exponential in the number of size of the original NFSFA (Mohri et al.,
 4342 2002).

4343 9.1.2 Morphology as a regular language

4344 Many words have internal structure, such as prefixes and suffixes that shape their mean-
 4345 ing. The study of word-internal structure is the domain of **morphology**, of which there
 4346 are two main types:

- 4347 • **Derivational morphology** describes the use of affixes to convert a word from one
 4348 grammatical category to another (e.g., from the noun *grace* to the adjective *graceful*),
 4349 or to change the meaning of the word (e.g., from *grace* to *disgrace*).
- 4350 • **Inflectional morphology** describes the addition of details such as gender, number,
 4351 person, and tense (e.g., the *-ed* suffix for past tense in English).

4352 Morphology is a rich topic in linguistics, deserving of a course in its own right.¹ The
 4353 focus here will be on the use of finite state automata for morphological analysis. The

¹A good starting point would be a chapter from a linguistics textbook (e.g., Akmajian et al., 2010; Bender, 2013). A key simplification in this chapter is the focus on affixes at the sole method of derivation and inflection. English makes use of affixes, but also incorporates **apophony**, such as the inflection of *foot* to *feet*. Semitic languages like Arabic and Hebrew feature a template-based system of morphology, in which roots are triples of consonants (e.g., *ktb*), and words are created by adding vowels: *kataba* (Arabic: he wrote), *kutub* (books), *maktab* (desk). For more detail on morphology, see texts from Haspelmath and Sims (2013) and Lieber (2015).

4354 current section deals with derivational morphology; inflectional morphology is discussed
 4355 in § 9.1.4.3.

4356 Suppose that we want to write a program that accepts only those words that are con-
 4357 structed in accordance with the rules of English derivational morphology:

- 4358 (9.1) grace, graceful, gracefully, *gracelyful
- 4359 (9.2) disgrace, *ungrace, disgraceful, disgracefully
- 4360 (9.3) allure, *allureful, alluring, alluringly
- 4361 (9.4) fairness, unfair, *disfair, fairly

4362 (Recall that the asterisk indicates that a linguistic example is judged unacceptable by flu-
 4363 ent speakers of a language.) These examples cover only a tiny corner of English deriva-
 4364 tional morphology, but a number of things stand out. The suffix *-ful* converts the nouns
 4365 *grace* and *disgrace* into adjectives, and the suffix *-ly* converts adjectives into adverbs. These
 4366 suffixes must be applied in the correct order, as shown by the unacceptability of **grace-*
 4367 *lyful*. The *-ful* suffix works for only some words, as shown by the use of *alluring* as the
 4368 adjectival form of *allure*. Other changes are made with prefixes, such as the derivation
 4369 of *disgrace* from *grace*, which roughly corresponds to a negation; however, *fair* is negated
 4370 with the *un-* prefix instead. Finally, while the first three examples suggest that the direc-
 4371 tion of derivation is noun → adjective → adverb, the example of *fair* suggests that the
 4372 adjective can also be the base form, with the *-ness* suffix performing the conversion to a
 4373 noun.

4374 Can we build a computer program that accepts only well-formed English words, and
 4375 rejects all others? This might at first seem trivial to solve with a brute-force attack: simply
 4376 make a dictionary of all valid English words. But such an approach fails to account for
 4377 morphological **productivity** — the applicability of existing morphological rules to new
 4378 words and names, such as *Trump* to *Trumpy* and *Trumpkin*, and *Clinton* to *Clintonian* and
 4379 *Clintonite*. We need an approach that represents morphological rules explicitly, and for
 4380 this we will try a finite state acceptor.

4381 The dictionary approach can be implemented as a finite state acceptor, with the vo-
 4382 cabulary Σ equal to the vocabulary of English, and a transition from the start state to the
 4383 accepting state for each word. But this would of course fail to generalize beyond the origi-
 4384 nal vocabulary, and would not capture anything about the **morphotactic** rules that govern
 4385 derivations from new words. The first step towards a more general approach is shown in
 4386 Figure 9.2, which is the state diagram for a finite state acceptor in which the vocabulary
 4387 consists of **morphemes**, which include **stems** (e.g., *grace*, *allure*) and **affixes** (e.g., *dis-*, *-ing*,
 4388 *-ly*). This finite state acceptor consists of a set of paths leading away from the start state,
 4389 with derivational affixes added along the path. Except for q_{neg} , the states on these paths
 4390 are all final, so the FSA will accept *disgrace*, *disgraceful*, and *disgracefully*, but not *dis-*.

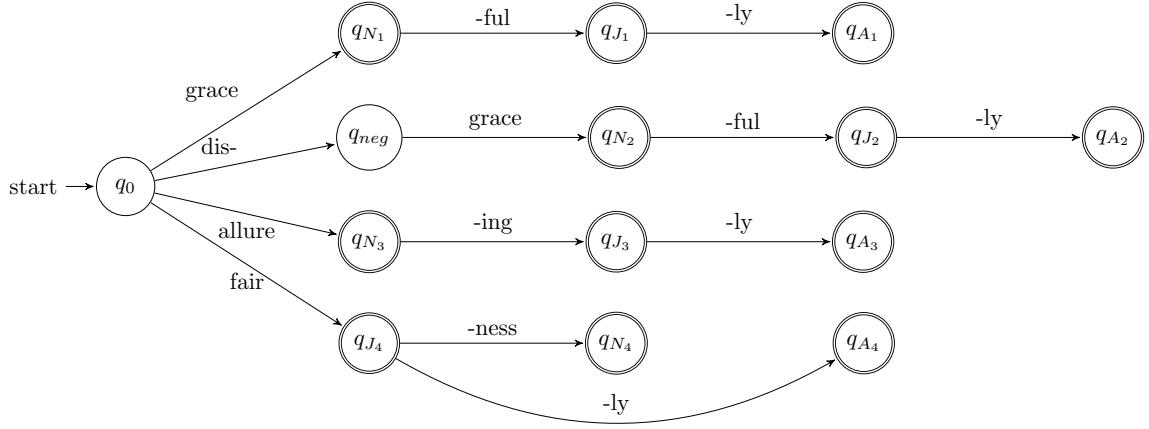


Figure 9.2: A finite state acceptor for a fragment of English derivational morphology. Each path represents possible derivations from a single root form.

4391 This FSA can be **minimized** to the form shown in Figure 9.3, which makes the generality of the finite state approach more apparent. For example, the transition from q_0 to
 4392 q_{J_2} can be made to accept not only *fair* but any single-morpheme (**monomorphemic**) adjective that takes *-ness* and *-ly* as suffixes. In this way, the finite state acceptor can easily
 4393 be extended: as new word stems are added to the vocabulary, their derived forms will be
 4394 accepted automatically. Of course, this FSA would still need to be extended considerably
 4395 to cover even this small fragment of English morphology. As shown by cases like *music*
 4396 → *musical*, *athlete* → *athletic*, English includes several classes of nouns, each with its own
 4397 rules for derivation.

4400 The FSAs shown in Figure 9.2 and 9.3 accept *allureing*, not *alluring*. This reflects a distinction between morphology — the question of which morphemes to use, and in what order — and **orthography** — the question of how the morphemes are rendered in written language. Just as orthography requires dropping the *e* preceding the *-ing* suffix, **phonology** imposes a related set of constraints on how words are rendered in speech. As we will see soon, these issues are handled through **finite state transducers**, which are finite state automata that take inputs and produce outputs.

4407 9.1.3 Weighted finite state acceptors

4408 According to the FSA treatment of morphology, every word is either in or out of the language, with no wiggle room. Perhaps you agree that *musicky* and *fishful* are not valid
 4409 English words; but if forced to choose, you probably find *a fishful stew* or *a musicky trib-
 4410 ute* preferable to *behaving disgracelyful*. Rather than asking whether a word is acceptable,
 4411 we might like to ask how acceptable it is. Aronoff (1976, page 36) puts it another way:

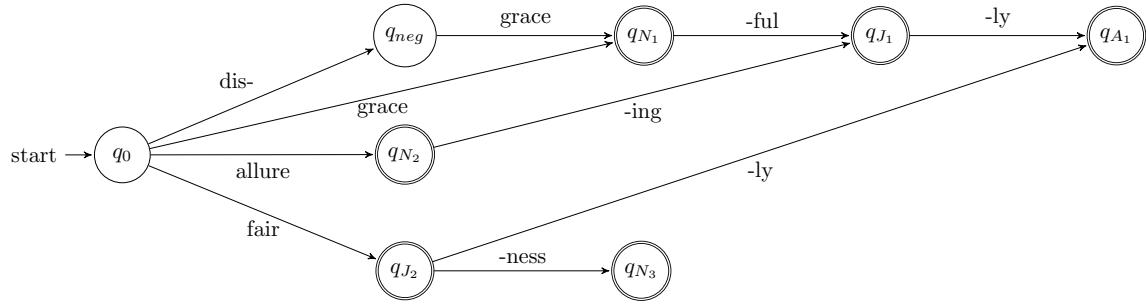


Figure 9.3: Minimization of the finite state acceptor shown in Figure 9.2.

4413 "Though many things are possible in morphology, some are more possible than others."
 4414 But finite state acceptors give no way to express preferences among technically valid
 4415 choices.

4416 **Weighted finite state acceptors (WFSAs)** are generalizations of FSAs, in which each
 4417 accepting path is assigned a score, computed from the transitions, the initial state, and the
 4418 final state. Formally, a weighted finite state acceptor $M = (Q, \Sigma, \lambda, \rho, \delta)$ consists of:

- 4419 • a finite set of states $Q = \{q_0, q_1, \dots, q_n\}$;
- 4420 • a finite alphabet Σ of input symbols;
- 4421 • an initial weight function, $\lambda : Q \mapsto \mathbb{R}$;
- 4422 • a final weight function $\rho : Q \mapsto \mathbb{R}$;
- 4423 • a transition function $\delta : Q \times \Sigma \times Q \mapsto \mathbb{R}$.

4424 WFSAs depart from the FSA formalism in three ways: every state can be an initial
 4425 state, with score $\lambda(q)$; every state can be an accepting state, with score $\rho(q)$; transitions are
 4426 possible between any pair of states on any input, with a score $\delta(q_i, \omega, q_j)$. Nonetheless,
 4427 FSAs can be viewed as a special case: for any FSA M we can build an equivalent WFSA
 4428 by setting $\lambda(q) = \infty$ for all $q \neq q_0$, $\rho(q) = \infty$ for all $q \notin F$, and $\delta(q_i, \omega, q_j) = \infty$ for all
 4429 transitions $\{(q_1, \omega) \rightarrow q_2\}$ that are not permitted by the transition function of M .

4430 The total score for any path $\pi = t_1, t_2, \dots, t_N$ is equal to the sum of these scores,

$$d(\pi) = \lambda(\text{from-state}(t_1)) + \sum_n^N \delta(t_n) + \rho(\text{to-state}(t_N)). \quad [9.5]$$

4431 A **shortest-path algorithm** is used to find the minimum-cost path through a WFSA for
 4432 string ω , with time complexity $\mathcal{O}(E + V \log V)$, where E is the number of edges and V is
 4433 the number of vertices (Cormen et al., 2009).²

²Shortest-path algorithms find the path with the minimum cost. In many cases, the path weights are log

4434 **9.1.3.1 N-gram language models as WFSAs**

4435 In **n-gram language models** (see § 6.1), the probability of a sequence of tokens w_1, w_2, \dots, w_M
 4436 is modeled as,

$$p(w_1, \dots, w_M) \approx \prod_{m=1}^M p_n(w_m | w_{m-1}, \dots, w_{m-n+1}). \quad [9.6]$$

The log probability under an n -gram language model can be modeled in a WFSA. First consider a unigram language model. We need only a single state q_0 , with transition scores $\delta(q_0, \omega, q_0) = \log p_1(\omega)$. The initial and final scores can be set to zero. Then the path score for w_1, w_2, \dots, w_M is equal to,

$$0 + \sum_m^M \delta(q_0, w_m, q_0) + 0 = \sum_m^M \log p_1(w_m). \quad [9.7]$$

For an n -gram language model with $n > 1$, we need probabilities that condition on the past history. For example, in a bigram language model, the transition weights must represent $\log p_2(w_m | w_{m-1})$. The transition scoring function must somehow “remember” the previous word or words. This can be done by adding more states: to model the bigram probability $p_2(w_m | w_{m-1})$, we need a state for every possible w_{m-1} — a total of V states. The construction indexes each state q_i by a context event $w_{m-1} = i$. The weights are then assigned as follows:

$$\begin{aligned} \delta(q_i, \omega, q_j) &= \begin{cases} \log \Pr(w_m = j | w_{m-1} = i), & \omega = j \\ -\infty, & \omega \neq j \end{cases} \\ \lambda(q_i) &= \log \Pr(w_1 = i | w_0 = \square) \\ \rho(q_i) &= \log \Pr(w_{M+1} = \blacksquare | w_M = i). \end{aligned}$$

4437 The transition function is designed to ensure that the context is recorded accurately:
 4438 we can move to state j on input ω only if $\omega = j$; otherwise, transitioning to state j is
 4439 forbidden by the weight of $-\infty$. The initial weight function $\lambda(q_i)$ is the log probability of
 4440 receiving i as the first token, and the final weight function $\rho(q_i)$ is the log probability of
 4441 receiving an “end-of-string” token after observing $w_M = i$.

4442 **9.1.3.2 *Semiring weighted finite state acceptors**

4443 The n -gram language model WFSA is deterministic: each input has exactly one accepting
 4444 path, for which the WFSA computes a score. In non-deterministic WFSAs, a given input

probabilities, so we want the path with the maximum score, which can be accomplished by making each local score into a *negative* log-probability. The remainder of this section will refer to **best-path algorithms**, which are assumed to “do the right thing.”

may have multiple accepting paths. In some applications, the score for the input is aggregated across all such paths. Such aggregate scores can be computed by generalizing WFSAs with **semiring notation**, first introduced in § 7.7.3.

Let $d(\pi)$ represent the total score for path $\pi = t_1, t_2, \dots, t_N$, which is computed as,

$$d(\pi) = \lambda(\text{from-state}(t_1)) \otimes \delta(t_1) \otimes \delta(t_2) \otimes \dots \otimes \delta(t_N) \otimes \rho(\text{to-state}(t_N)). \quad [9.8]$$

This is a generalization of Equation 9.5 to semiring notation, using the semiring multiplication operator \otimes in place of addition.

Now let $s(\omega)$ represent the total score for all paths $\Pi(\omega)$ that consume input ω ,

$$s(\omega) = \bigoplus_{\pi \in \Pi(\omega)} d(\pi). \quad [9.9]$$

Here, semiring addition (\oplus) is used to combine the scores of multiple paths.

The generalization to semirings covers a number of useful special cases. In the log-probability semiring, multiplication is defined as $\log p(x) \otimes \log p(y) = \log p(x) + \log p(y)$, and addition is defined as $\log p(x) \oplus \log p(y) = \log(p(x) + p(y))$. Thus, $s(\omega)$ represents the log-probability of accepting input ω , marginalizing over all paths $\pi \in \Pi(\omega)$. In the boolean semiring, the \otimes operator is logical conjunction, and the \oplus operator is logical disjunction. This reduces to the special case of unweighted finite state acceptors, where the score $s(\omega)$ is a boolean indicating whether there exists any accepting path for ω . In the tropical semiring, the \oplus operator is a maximum, so the resulting score is the score of the best-scoring path through the WFSAs. The OpenFST toolkit uses semirings and polymorphism to implement general algorithms for weighted finite state automata (Allauzen et al., 2007).

9.1.3.3 *Interpolated n -gram language models

Recall from § 6.2.3 that an **interpolated n -gram language model** combines the probabilities from multiple n -gram models. For example, an interpolated bigram language model computes probability,

$$\hat{p}(w_m | w_{m-1}) = \lambda_1 p_1(w_m) + \lambda_2 p_2(w_m | w_{m-1}), \quad [9.10]$$

with \hat{p} indicating the interpolated probability, p_2 indicating the bigram probability, and p_1 indicating the unigram probability. We set $\lambda_2 = (1 - \lambda_1)$ so that the probabilities sum to one.

Interpolated bigram language models can be implemented using a non-deterministic WFSAs (Knight and May, 2009). The basic idea is shown in Figure 9.4. In an interpolated bigram language model, there is one state for each element in the vocabulary — in this

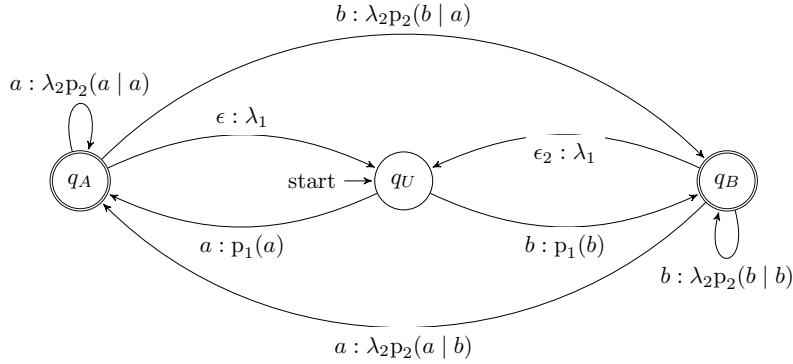


Figure 9.4: WFSA implementing an interpolated bigram/unigram language model, on the alphabet $\Sigma = \{a, b\}$. For simplicity, the WFSA is constrained to force the first token to be generated from the unigram model, and does not model the emission of the end-of-sequence token.

4474 case, the states q_A and q_B — which capture the contextual conditioning in the bigram
 4475 probabilities. To model unigram probabilities, there is an additional state q_U , which “forgets”
 4476 the context. Transitions out of q_U involve unigram probabilities, $p_1(a)$ and $p_2(b)$;
 4477 transitions into q_U emit the empty symbol ϵ , and have probability λ_1 , reflecting the inter-
 4478 polation weight for the unigram model. The interpolation weight for the bigram model is
 4479 included in the weight of the transition $q_A \rightarrow q_B$.

4480 The epsilon transitions into q_U make this WFSA non-deterministic. Consider the score
 4481 for the sequence (a, b, b) . The initial state is q_U , so the symbol a is generated with score
 4482 $p_1(a)$ ³ Next, we can generate b from the unigram model by taking the transition $q_A \rightarrow q_B$,
 4483 with score $\lambda_2 p_2(b | a)$. Alternatively, we can take a transition back to q_U with score λ_1 ,
 4484 and then emit b from the unigram model with score $p_1(b)$. To generate the final b token,
 4485 we face the same choice: emit it directly from the self-transition to q_B , or transition to q_U
 4486 first.

The total score for the sequence (a, b, b) is the semiring sum over all accepting paths,

$$\begin{aligned}
 s(a, b, b) &= (p_1(a) \otimes \lambda_2 p_2(b | a) \otimes \lambda_2 p_2(b | b)) \\
 &\oplus (p_1(a) \otimes \lambda_1 \otimes p_1(b) \otimes \lambda_2 p_2(b | b)) \\
 &\oplus (p_1(a) \otimes \lambda_2 p_2(b | a) \otimes p_1(b) \otimes p_1(b)) \\
 &\oplus (p_1(a) \otimes \lambda_1 \otimes p_1(b) \otimes p_1(b) \otimes p_1(b)). \tag{[9.11]}
 \end{aligned}$$

4487 Each line in Equation 9.11 represents the probability of a specific path through the WFSA.
 4488 In the probability semiring, \otimes is multiplication, so that each path is the product of each

³We could model the sequence-initial bigram probability $p_2(a | \square)$, but for simplicity the WFSA does not admit this possibility, which would require another state.

4489 transition weight, which are themselves probabilities. The \oplus operator is addition, so that
 4490 the total score is the sum of the scores (probabilities) for each path. This corresponds to
 4491 the probability under the interpolated bigram language model.

4492 **9.1.4 Finite state transducers**

4493 Finite state acceptors can determine whether a string is in a regular language, and weighted
 4494 finite state acceptors can compute a score for every string over a given alphabet. **Finite**
 4495 **state transducers** (FSTs) extend the formalism further, by adding an output symbol to each
 4496 transition. Formally, a finite state transducer is a tuple $T = (Q, \Sigma, \Omega, \lambda, \rho, \delta)$, with Ω repre-
 4497 senting an output vocabulary and the transition function $\delta : Q \times (\Sigma \cup \epsilon) \times (\Omega \cup \epsilon) \times Q \rightarrow \mathbb{R}$
 4498 mapping from states, input symbols, and output symbols to states. The remaining ele-
 4499 ments (Q, Σ, λ, ρ) are identical to their definition in weighted finite state acceptors (§ 9.1.3).
 4500 Thus, each path through the FST T transduces the input string into an output.

4501 **9.1.4.1 String edit distance**

The **edit distance** between two strings s and t is a measure of how many operations are required to transform one string into another. There are several ways to compute edit distance, but one of the most popular is the **Levenshtein edit distance**, which counts the minimum number of insertions, deletions, and substitutions. This can be computed by a one-state weighted finite state transducer, in which the input and output alphabets are identical. For simplicity, consider the alphabet $\Sigma = \Omega = \{a, b\}$. The edit distance can be computed by a one-state transducer with the following transitions,

$$\delta(q, a, a, q) = \delta(q, b, b, q) = 0 \quad [9.12]$$

$$\delta(q, a, b, q) = \delta(q, b, a, q) = 1 \quad [9.13]$$

$$\delta(q, a, \epsilon, q) = \delta(q, b, \epsilon, q) = 1 \quad [9.14]$$

$$\delta(q, \epsilon, a, q) = \delta(q, \epsilon, b, q) = 1. \quad [9.15]$$

4502 The state diagram is shown in Figure 9.5.

4503 For a given string pair, there are multiple paths through the transducer: the best-
 4504 scoring path from *dessert* to *desert* involves a single deletion, for a total score of 1; the
 4505 worst-scoring path involves seven deletions and six additions, for a score of 13.

4506 **9.1.4.2 The Porter stemmer**

The Porter (1980) stemming algorithm is a “lexicon-free” algorithm for stripping suffixes from English words, using a sequence of character-level rules. Each rule can be described

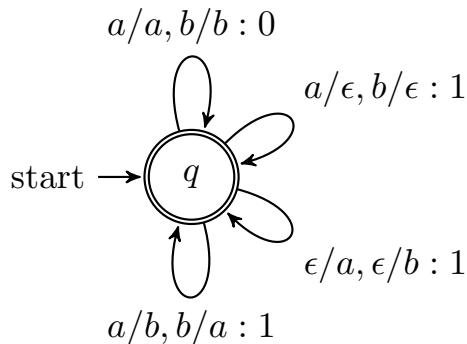


Figure 9.5: State diagram for the Levenshtein edit distance finite state transducer. The label $x/y : c$ indicates a cost of c for a transition with input x and output y .

by an unweighted finite state transducer. The first rule is:

-sses → -ss e.g., *dresses* → *dress* [9.16]

-ies → -i e.g., *parties* → *parti* [9.17]

-ss → -ss e.g., *dress* → *dress* [9.18]

-s → ε e.g., *cats* → *cat* [9.19]

4507 The final two lines appear to conflict; they are meant to be interpreted as an instruction
 4508 to remove a terminal *-s* unless it is part of an *-ss* ending. A state diagram to handle just
 4509 these final two lines is shown in Figure 9.6. Make sure you understand how this finite
 4510 state transducer handles *cats*, *steps*, *bass*, and *basses*.

4511 9.1.4.3 Inflectional morphology

4512 In **inflectional morphology**, word **lemmas** are modified to add grammatical information
 4513 such as tense, number, and case. For example, many English nouns are pluralized by the
 4514 suffix *-s*, and many verbs are converted to past tense by the suffix *-ed*. English's inflectional
 4515 morphology is considerably simpler than many of the world's languages. For example,
 4516 Romance languages (derived from Latin) feature complex systems of verb suffixes which
 4517 must agree with the person and number of the verb, as shown in Table 9.1.

4518 The task of **morphological analysis** is to read a form like *canto*, and output an analysis
 4519 like CANTAR+VERB+PRESIND+1P+SING, where +PRESIND describes the tense as present
 4520 indicative, +1P indicates the first-person, and +SING indicates the singular number. The
 4521 task of **morphological generation** is the reverse, going from CANTAR+VERB+PRESIND+1P+SING
 4522 to *canto*. Finite state transducers are an attractive solution, because they can solve both
 4523 problems with a single model (Beesley and Karttunen, 2003). As an example, Figure 9.7
 4524 shows a fragment of a finite state transducer for Spanish inflectional morphology. The

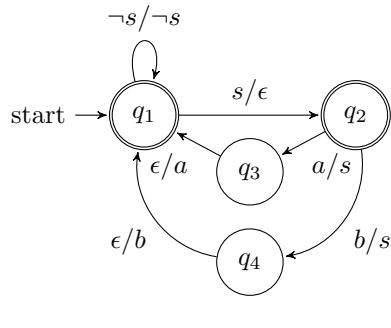


Figure 9.6: State diagram for final two lines of step 1a of the Porter stemming diagram. States q_3 and q_4 “remember” the observations a and b respectively; the ellipsis \dots represents additional states for each symbol in the input alphabet. The notation $\neg s / \neg s$ is not part of the FST formalism; it is a shorthand to indicate a set of self-transition arcs for every input/output symbol except s .

infinitive	cantar (to sing)	comer (to eat)	vivir (to live)
yo (1st singular)	canto	como	vivo
tu (2nd singular)	cantas	comes	vives
él, ella, usted (3rd singular)	canta	come	vive
nosotros (1st plural)	cantamos	comemos	vivimos
vosotros (2nd plural, informal)	cantáis	coméis	vívís
ellos, ellas (3rd plural); ustedes (2nd plural)	cantan	comen	viven

Table 9.1: Spanish verb inflections for the present indicative tense. Each row represents a person and number, and each column is a regular example from a class of verbs, as indicated by the ending of the infinitive form.

4525 input vocabulary Σ corresponds to the set of letters used in Spanish spelling, and the out-
 4526 put vocabulary Ω corresponds to these same letters, plus the vocabulary of morphological
 4527 features (e.g., +SING, +VERB). In Figure 9.7, there are two paths that take *canto* as input,
 4528 corresponding to the verb and noun meanings; the choice between these paths could be
 4529 guided by a part-of-speech tagger. By **inversion**, the inputs and outputs for each trans-
 4530ition are switched, resulting in a finite state generator, capable of producing the correct
 4531 **surface form** for any morphological analysis.

4532 Finite state morphological analyzers and other unweighted transducers can be de-
 4533 signed by hand. The designer’s goal is to avoid **overgeneration** — accepting strings or
 4534 making transductions that are not valid in the language — as well as **undergeneration** —

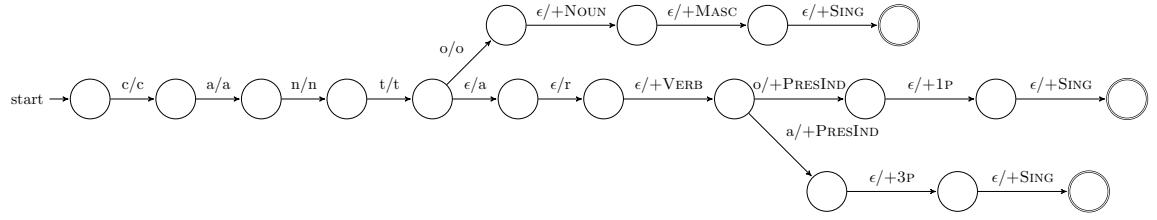


Figure 9.7: Fragment of a finite state transducer for Spanish morphology. There are two accepting paths for the input *canto*: *canto+NOUN+MASC+SING* (masculine singular noun, meaning a song), and *cantar+VERB+PRESIND+1P+SING* (I sing). There is also an accepting path for *canta*, with output *cantar+VERB+PRESIND+3P+SING* (he/she sings).

4535 failing to accept strings or transductions that are valid. For example, a pluralization trans-
 4536 ducer that does not accept *foot/feet* would undergenerate. Suppose we “fix” the transducer
 4537 to accept this example, but as a side effect, it now accepts *boot/beet*; the transducer would
 4538 then be said to overgenerate. A transducer that accepts *foot/foots* but not *foot/feet* would
 4539 both overgenerate and undergenerate.

4540 9.1.4.4 Finite state composition

4541 Designing finite state transducers to capture the full range of morphological phenomena
 4542 in any real language is a huge task. Modularization is a classic computer science approach
 4543 for this situation: decompose a large and unwieldy problem into a set of subproblems,
 4544 each of which will hopefully have a concise solution. Finite state automata can be mod-
 4545 ularized through **composition**: feeding the output of one transducer T_1 as the input to
 4546 another transducer T_2 , written $T_2 \circ T_1$. Formally, if there exists some y such that $(x, y) \in T_1$
 4547 (meaning that T_1 produces output y on input x), and $(y, z) \in T_2$, then $(x, z) \in (T_2 \circ T_1)$.
 4548 Because finite state transducers are closed under composition, there is guaranteed to be
 4549 a single finite state transducer that $T_3 = T_2 \circ T_1$, which can be constructed as a machine
 4550 with one state for each pair of states in T_1 and T_2 (Mohri et al., 2002).

4551 **Example: Morphology and orthography** In English morphology, the suffix *-ed* is added
 4552 to signal the past tense for many verbs: *cook*→*cooked*, *want*→*wanted*, etc. However, English
 4553 **orthography** dictates that this process cannot produce a spelling with consecutive *e*'s, so
 4554 that *bake*→*baked*, not *bakeed*. A modular solution is to build separate transducers for mor-
 4555 phology and orthography. The morphological transducer T_M transduces from *bake+PAST*
 4556 to *bake+ed*, with the *+* symbol indicating a segment boundary. The input alphabet of T_M
 4557 includes the lexicon of words and the set of morphological features; the output alphabet
 4558 includes the characters *a-z* and the *+* boundary marker. Next, an orthographic transducer
 4559 T_O is responsible for the transductions *cook+ed*→*cooked*, and *bake+ed*→*baked*. The input
 4560 alphabet of T_O must be the same as the output alphabet for T_M , and the output alphabet

4561 is simply the characters *a-z*. The composed transducer ($T_O \circ T_M$) then transduces from
 4562 *bake*+PAST to the spelling *baked*. The design of T_O is left as an exercise.

Example: Hidden Markov models Hidden Markov models (chapter 7) can be viewed as weighted finite state transducers, and they can be constructed by transduction. Recall that a hidden Markov model defines a joint probability over words and tags, $p(w, y)$, which can be computed as a path through a **trellis** structure. This trellis is itself a weighted finite state acceptor, with edges between all adjacent nodes $q_{m-1,i} \rightarrow q_{m,j}$ on input $Y_m = j$. The edge weights are log-probabilities,

$$\delta(q_{m-1,i}, Y_m = j, q_{m,j}) = \log p(w_m, Y_m = j | Y_{m-1} = i) \quad [9.20]$$

$$= \log p(w_m | Y_m = j) + \log \Pr(Y_m = j | Y_{m-1} = i). \quad [9.21]$$

4563 Because there is only one possible transition for each tag Y_m , this WFSA is deterministic.
 4564 The score for any tag sequence $\{y_m\}_{m=1}^M$ is the sum of these log-probabilities, correspond-
 4565 ing to the total log probability $\log p(w, y)$. Furthermore, the trellis can be constructed by
 4566 the composition of simpler FSTs.

- 4567 • First, construct a “transition” transducer to represent a bigram probability model
 4568 over tag sequences, T_T . This transducer is almost identical to the n -gram language
 4569 model acceptor in § 9.1.3.1: there is one state for each tag, and the edge weights
 4570 equal to the transition log-probabilities, $\delta(q_i, j, j, q_j) = \log \Pr(Y_m = j | Y_{m-1} = i)$.
 4571 Note that T_T is a transducer, with identical input and output at each arc; this makes
 4572 it possible to compose T_T with other transducers.
- 4573 • Next, construct an “emission” transducer to represent the probability of words given
 4574 tags, T_E . This transducer has only a single state, with arcs for each word/tag pair,
 4575 $\delta(q_0, i, j, q_0) = \log \Pr(W_m = j | Y_m = i)$. The input vocabulary is the set of all tags,
 4576 and the output vocabulary is the set of all words.
- 4577 • The composition $T_E \circ T_T$ is a finite state transducer with one state per tag, as shown
 4578 in Figure 9.8. Each state has $V \times K$ outgoing edges, representing transitions to each
 4579 of the K other states, with outputs for each of the V words in the vocabulary. The
 4580 weights for these edges are equal to,

$$\delta(q_i, Y_m = j, w_m, q_j) = \log p(w_m, Y_m = j | Y_{m-1} = i). \quad [9.22]$$

- 4581 • The trellis is a structure with $M \times K$ nodes, for each of the M words to be tagged and
 4582 each of the K tags in the tagset. It can be built by composition of $(T_E \circ T_T)$ against an
 4583 unweighted **chain FSA** $M_A(w)$ that is specially constructed to accept only a given
 4584 input w_1, w_2, \dots, w_M , shown in Figure 9.9. The trellis for input w is built from the
 4585 composition $M_A(w) \circ (T_E \circ T_T)$. Composing with the unweighted $M_A(w)$ does not
 4586 affect the edge weights from $(T_E \circ T_T)$, but it selects the subset of paths that generate
 4587 the word sequence w .

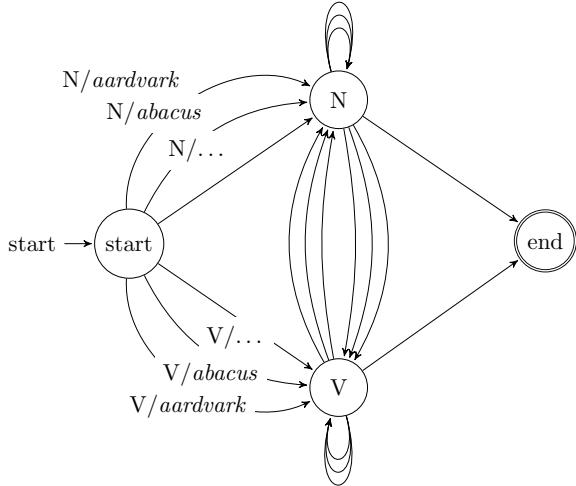


Figure 9.8: Finite state transducer for hidden Markov models, with a small tagset of nouns and verbs. For each pair of tags (including self-loops), there is an edge for every word in the vocabulary. For simplicity, input and output are only shown for the edges from the start state. Weights are also omitted from the diagram; for each edge from q_i to q_j , the weight is equal to $\log p(w_m, Y_m = j \mid Y_{m-1} = i)$, except for edges to the end state, which are equal to $\log \Pr(Y_m = \diamond \mid Y_{m-1} = i)$.

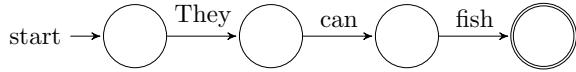


Figure 9.9: Chain finite state acceptor for the input *They can fish*.

4588 9.1.5 *Learning weighted finite state automata

4589 In generative models such as n -gram language models and hidden Markov models, the
 4590 edge weights correspond to log probabilities, which can be obtained from relative fre-
 4591 quency estimation. However, in other cases, we wish to learn the edge weights from in-
 4592 put/output pairs. This is difficult in non-deterministic finite state automata, because we
 4593 do not observe the specific arcs that are traversed in accepting the input, or in transducing
 4594 from input to output. The path through the automaton is a **latent variable**.

4595 Chapter 5 presented one method for learning with latent variables: expectation max-
 4596 imization (EM). This involves computing a distribution $q(\cdot)$ over the latent variable, and
 4597 iterating between updates to this distribution and updates to the parameters — in this
 4598 case, the arc weights. The **forward-backward algorithm** (§ 7.5.3.3) describes a dynamic
 4599 program for computing a distribution over arcs in the trellis structure of a hidden Markov

model, but this is a special case of the more general problem for finite state automata. Eisner (2002) describes an **expectation semiring**, which enables the expected number of transitions across each arc to be computed through a semiring shortest-path algorithm. Alternative approaches for generative models include Markov Chain Monte Carlo (Chiang et al., 2010) and spectral learning (Balle et al., 2011).

Further afield, we can take a perceptron-style approach, with each arc corresponding to a feature. The classic perceptron update would update the weights by subtracting the difference between the feature vector corresponding to the predicted path and the feature vector corresponding to the correct path. Since the path is not observed, we resort to a **hidden variable perceptron**. The model is described formally in § 12.4, but the basic idea is to compute an update from the difference between the features from the predicted path and the features for the best-scoring path that generates the correct output.

9.2 Context-free languages

Beyond the class of regular languages lie the context-free languages. An example of a language that is context-free but not finite state is the set of arithmetic expressions with balanced parentheses. Intuitively, to accept only strings in this language, an FSA would have to “count” the number of left parentheses, and make sure that they are balanced against the number of right parentheses. An arithmetic expression can be arbitrarily long, yet by definition an FSA has a finite number of states. Thus, for any FSA, there will be a string that with too many parentheses to count. More formally, the **pumping lemma** is a proof technique for showing that languages are not regular. It is typically demonstrated for the simpler case $a^n b^n$, the language of strings containing a sequence of a 's, and then an equal-length sequence of b 's.⁴

There are at least two arguments for the relevance of non-regular formal languages to linguistics. First, there are natural language phenomena that are argued to be isomorphic to $a^n b^n$. For English, the classic example is **center embedding**, shown in Figure 9.10. The initial expression *the dog* specifies a single dog. Embedding this expression into *the cat ... chased* specifies a particular cat — the one chased by the dog. This cat can then be embedded again to specify a goat, in the less felicitous but arguably grammatical expression, *the goat the cat the dog chased kissed*, which refers to the goat who was kissed by the cat which was chased by the dog. Chomsky (1957) argues that to be grammatical, a center-embedded construction must be balanced: if it contains n noun phrases (e.g., *the cat*), they must be followed by exactly $n - 1$ verbs. An FSA that could recognize such expressions would also be capable of recognizing the language $a^n b^n$. Because we can prove that no FSA exists for $a^n b^n$, no FSA can exist for center embedded constructions either. En-

⁴Details of the proof can be found in an introductory computer science theory textbook (e.g., Sipser, 2012).

			the dog	
	the cat	the dog	chased	
the goat	the cat	the dog	chased	kissed
			...	

Figure 9.10: Three levels of center embedding

4635 glish includes center embedding, and so the argument goes, English grammar as a whole
 4636 cannot be regular.⁵

4637 A more practical argument for moving beyond regular languages is modularity. Many
 4638 linguistic phenomena — especially in syntax — involve constraints that apply at long
 4639 distance. Consider the problem of determiner-noun number agreement in English: we
 4640 can say *the coffee* and *these coffees*, but not **these coffee*. By itself, this is easy enough to model
 4641 in an FSA. However, fairly complex modifying expressions can be inserted between the
 4642 determiner and the noun:

- 4643 (9.5) the burnt coffee
- 4644 (9.6) the badly-ground coffee
- 4645 (9.7) the burnt and badly-ground Italian coffee
- 4646 (9.8) these burnt and badly-ground Italian coffees
- 4647 (9.9) *these burnt and badly-ground Italian coffee

4648 Again, an FSA can be designed to accept modifying expressions such as *burnt and badly-*
 4649 *ground Italian*. Let's call this FSA F_M . To reject the final example, a finite state acceptor
 4650 must somehow "remember" that the determiner was plural when it reaches the noun *cof-*
 4651 *fee* at the end of the expression. The only way to do this is to make two identical copies
 4652 of F_M : one for singular determiners, and one for plurals. While this is possible in the
 4653 finite state framework, it is inconvenient — especially in languages where more than one
 4654 attribute of the noun is marked by the determiner. **Context-free languages** facilitate mod-
 4655 ularity across such long-range dependencies.

4656 9.2.1 Context-free grammars

4657 Context-free languages are specified by **context-free grammars (CFGs)**, which are tuples
 4658 (N, Σ, R, S) consisting of:

⁵The claim that arbitrarily deep center-embedded expressions are grammatical has drawn skepticism. Corpus evidence shows that embeddings of depth greater than two are exceedingly rare (Karlsson, 2007), and that embeddings of depth greater than three are completely unattested. If center-embedding is capped at some finite depth, then it is regular.

$$\begin{aligned}
 S &\rightarrow S \text{ OP } S \mid \text{NUM} \\
 \text{OP} &\rightarrow + \mid - \mid \times \mid \div \\
 \text{NUM} &\rightarrow \text{NUM DIGIT} \mid \text{DIGIT} \\
 \text{DIGIT} &\rightarrow 0 \mid 1 \mid 2 \mid \dots \mid 9
 \end{aligned}$$

Figure 9.11: A context-free grammar for arithmetic expressions

- 4659 • a finite set of **non-terminals** N ;
 4660 • a finite alphabet Σ of **terminal symbols**;
 4661 • a set of **production rules** R , each of the form $A \rightarrow \beta$, where $A \in N$ and $\beta \in (\Sigma \cup N)^*$;
 4662 • a designated start symbol S .

4663 In the production rule $A \rightarrow \beta$, the left-hand side (LHS) A must be a non-terminal;
 4664 the right-hand side (RHS) can be a sequence of terminals or non-terminals, $\{n, \sigma\}^*, n \in$
 4665 $N, \sigma \in \Sigma$. A non-terminal can appear on the left-hand side of many production rules.
 4666 A non-terminal can appear on both the left-hand side and the right-hand side; this is a
 4667 **recursive production**, and is analogous to self-loops in finite state automata. The name
 4668 “context-free” is based on the property that the production rule depends only on the LHS,
 4669 and not on its ancestors or neighbors; this is analogous to Markov property of finite state
 4670 automata, in which the behavior at each step depends only on the current state, on not on
 4671 the path by which that state was reached.

4672 A **derivation** τ is a sequence of steps from the start symbol S to a surface string $w \in \Sigma^*$,
 4673 which is the **yield** of the derivation. A string w is in a context-free language if there is
 4674 some derivation from S yielding w . **Parsing** is the problem of finding a derivation for a
 4675 string in a grammar. Algorithms for parsing are described in chapter 10.

4676 Like regular expressions, context-free grammars define the language but not the com-
 4677 putation necessary to recognize it. The context-free analogues to finite state acceptors are
 4678 **pushdown automata**, a theoretical model of computation in which input symbols can be
 4679 pushed onto a stack with potentially infinite depth. For more details, see Sipser (2012).

4680 9.2.1.1 Example

4681 Figure 9.11 shows a context-free grammar for arithmetic expressions such as $1 + 2 \div 3 - 4$.
 4682 In this grammar, the terminal symbols include the digits $\{1, 2, \dots, 9\}$ and the op-
 4683 erators $\{+, -, \times, \div\}$. The rules include the $|$ symbol, a notational convenience that makes
 4684 it possible to specify multiple right-hand sides on a single line: the statement $A \rightarrow x | y$

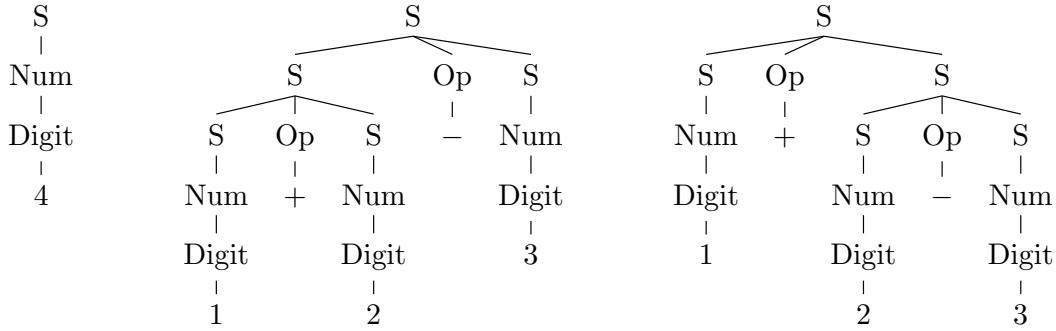


Figure 9.12: Some example derivations from the arithmetic grammar in Figure 9.11

4685 defines *two* productions, $A \rightarrow x$ and $A \rightarrow y$. This grammar is recursive: the non-termals S
4686 and NUM can produce themselves.

4687 Derivations are typically shown as trees, with production rules applied from the top
4688 to the bottom. The tree on the left in Figure 9.12 describes the derivation of a single digit,
4689 through the sequence of productions $S \rightarrow \text{NUM} \rightarrow \text{DIGIT} \rightarrow 4$ (these are all **unary produc-**
4690 **tions**, because the right-hand side contains a single element). The other two trees in
4691 Figure 9.12 show alternative derivations of the string $1 + 2 - 3$. The existence of multiple
4692 derivations for a string indicates that the grammar is **ambiguous**.

Context-free derivations can also be written out according to the pre-order tree traversal.⁶ For the two derivations of $1 + 2 - 3$ in Figure 9.12, the notation is:

$$(S (S (S (\text{Num} (Digit 1))) (\text{Op} +) (S (\text{Num} (Digit 2))))) (\text{Op} -) (S (\text{Num} (Digit 3)))) \quad [9.23]$$

$$(S (S (\text{Num} (Digit 1))) (\text{Op} +) (S (\text{Num} (Digit 2)) (\text{Op} -) (S (\text{Num} (Digit 3)))))). \quad [9.24]$$

4693 9.2.1.2 Grammar equivalence and Chomsky Normal Form

A single context-free language can be expressed by more than one context-free grammar. For example, the following two grammars both define the language $a^n b^n$ for $n > 0$.

$$\begin{aligned} S &\rightarrow aSb \mid ab \\ S &\rightarrow aSb \mid aabb \mid ab \end{aligned}$$

4694 Two grammars are **weakly equivalent** if they generate the same strings. Two grammars
4695 are **strongly equivalent** if they generate the same strings via the same derivations. The
4696 grammars above are only weakly equivalent.

⁶This is a depth-first left-to-right search that prints each node the first time it is encountered (Cormen et al., 2009, chapter 12).

In **Chomsky Normal Form (CNF)**, the right-hand side of every production includes either two non-terminals, or a single terminal symbol:

$$A \rightarrow BC$$

$$A \rightarrow a$$

4697 All CFGs can be converted into a CNF grammar that is weakly equivalent. To convert a
 4698 grammar into CNF, we first address productions that have more than two non-terminals
 4699 on the RHS by creating new “dummy” non-terminals. For example, if we have the pro-
 4700 duction,

$$W \rightarrow X Y Z, \quad [9.25]$$

it is replaced with two productions,

$$W \rightarrow X W \setminus X \quad [9.26]$$

$$W \setminus X \rightarrow Y Z. \quad [9.27]$$

4701 In these productions, $W \setminus X$ is a new dummy non-terminal. This transformation **binarizes**
 4702 the grammar, which is critical for efficient bottom-up parsing, as we will see in chapter 10.
 4703 Productions whose right-hand side contains a mix of terminal and non-terminal symbols
 4704 can be replaced in a similar fashion.

4705 Unary non-terminal productions $A \rightarrow B$ are replaced as follows: identify all produc-
 4706 tions $B \rightarrow \alpha$, and add $A \rightarrow \alpha$ to the grammar. For example, in the grammar described in
 4707 Figure 9.11, we would replace $\text{NUM} \rightarrow \text{DIGIT}$ with $\text{NUM} \rightarrow 1 \mid 2 \mid \dots \mid 9$. However, we
 4708 keep the production $\text{NUM} \rightarrow \text{NUM DIGIT}$, which is a valid binary production.

4709 9.2.2 Natural language syntax as a context-free language

4710 Context-free grammars are widely used to represent **syntax**, which is the set of rules that
 4711 determine whether an utterance is judged to be grammatical. If this representation were
 4712 perfectly faithful, then a natural language such as English could be transformed into a
 4713 formal language, consisting of exactly the (infinite) set of strings that would be judged to
 4714 be grammatical by a fluent English speaker. We could then build parsing software that
 4715 would automatically determine if a given utterance were grammatical.⁷

4716 Contemporary theories generally do *not* consider natural languages to be context-free
 4717 (see § 9.3), yet context-free grammars are widely used in natural language parsing. The
 4718 reason is that context-free representations strike a good balance: they cover a broad range
 4719 of syntactic phenomena, and they can be parsed efficiently. This section therefore de-
 4720 scribes how to handle a core fragment of English syntax in context-free form, following

⁷You are encouraged to move beyond this cursory treatment of syntax by consulting a textbook on linguistics (e.g., Akmajian et al., 2010; Bender, 2013).

4721 the conventions of the **Penn Treebank** (PTB; Marcus et al., 1993), a large-scale annotation
 4722 of English language syntax. The generalization to “mildly” context-sensitive languages is
 4723 discussed in § 9.3.

4724 The Penn Treebank annotation is a **phrase-structure grammar** of English. This means
 4725 that sentences are broken down into **constituents**, which are contiguous sequences of
 4726 words that function as coherent units for the purpose of linguistic analysis. Constituents
 4727 generally have a few key properties:

4728 **Movement.** Constituents can often be moved around sentences as units.

- 4729 (9.10) Abigail gave (her brother) (a fish).
 4730 (9.11) Abigail gave (a fish) to (her brother).

4731 In contrast, *gave her* and *brother a* cannot easily be moved while preserving gram-
 4732 maticality.

4733 **Substitution.** Constituents can be substituted by other phrases of the same type.

- 4734 (9.12) Max thanked (his older sister).
 4735 (9.13) Max thanked (her).

4736 In contrast, substitution is not possible for other contiguous units like *Max thanked*
 4737 and *thanked his*.

4738 **Coordination.** Coordinators like *and* and *or* can conjoin constituents.

- 4739 (9.14) (Abigail) and (her younger brother) bought a fish.
 4740 (9.15) Abigail (bought a fish) and (gave it to Max).
 4741 (9.16) Abigail (bought) and (greedily ate) a fish.

4742 Units like *brother bought* and *bought a* cannot easily be coordinated.

4743 These examples argue for units such as *her brother* and *bought a fish* to be treated as con-
 4744 stituents. Other sequences of words in these examples, such as *Abigail gave* and *brother*
a fish, cannot be moved, substituted, and coordinated in these ways. In phrase-structure
 4745 grammar, constituents are nested, so that *the senator from New Jersey* contains the con-
 4746 stituent *from New Jersey*, which in turn contains *New Jersey*. The sentence itself is the max-
 4747 imal constituent; each word is a minimal constituent, derived from a unary production
 4748 from a part-of-speech tag. Between part-of-speech tags and sentences are **phrases**. In
 4749 phrase-structure grammar, phrases have a type that is usually determined by their **head**
 4750 **word**: for example, a **noun phrase** corresponds to a noun and the group of words that

4752 modify it, such as *her younger brother*; a **verb phrase** includes the verb and its modifiers,
4753 such as *bought a fish* and *greedily ate it*.

4754 In context-free grammars, each phrase type is a non-terminal, and each constituent is
4755 the substring that the non-terminal yields. Grammar design involves choosing the right
4756 set of non-terminals. Fine-grained non-terminals make it possible to represent more fine-
4757 grained linguistic phenomena. For example, by distinguishing singular and plural noun
4758 phrases, it is possible to have a grammar of English that generates only sentences that
4759 obey subject-verb agreement. However, enforcing subject-verb agreement is considerably
4760 more complicated in languages like Spanish, where the verb must agree in both person
4761 and number with subject. In general, grammar designers must trade off between **over-**
4762 **generation** — a grammar that permits ungrammatical sentences — and **undergeneration**
4763 — a grammar that fails to generate grammatical sentences. Furthermore, if the grammar is
4764 to support manual annotation of syntactic structure, it must be simple enough to annotate
4765 efficiently.

4766 9.2.3 A phrase-structure grammar for English

4767 To better understand how phrase-structure grammar works, let's consider the specific
4768 case of the Penn Treebank grammar of English. The main phrase categories in the Penn
4769 Treebank (PTB) are based on the main part-of-speech classes: noun phrase (NP), verb
4770 phrase (VP), prepositional phrase (PP), adjectival phrase (ADJP), and adverbial phrase
4771 (ADVP). The top-level category is S, which conveniently stands in for both "sentence"
4772 and the "start" symbol. **Complement clauses** (e.g., *I take the good old fashioned ground that*
4773 *the whale is a fish*) are represented by the non-terminal SBAR. The terminal symbols in
4774 the grammar are individual words, which are generated from unary productions from
4775 part-of-speech tags (the PTB tagset is described in § 8.1).

4776 This section explores the productions from the major phrase-level categories, explaining
4777 how to generate individual tag sequences. The production rules are approached in a
4778 "theory-driven" manner: first the syntactic properties of each phrase type are described,
4779 and then some of the necessary production rules are listed. But it is important to keep
4780 in mind that the Penn Treebank was produced in a "data-driven" manner. After the set
4781 of non-terminals was specified, annotators were free to analyze each sentence in what-
4782 ever way seemed most linguistically accurate, subject to some high-level guidelines. The
4783 grammar of the Penn Treebank is simply the set of productions that were required to ana-
4784 lyze the several million words of the corpus. By design, the grammar overgenerates — it
4785 does not exclude ungrammatical sentences.

4786 **9.2.3.1 Sentences**

The most common production rule for sentences is,

$$S \rightarrow NP VP \quad [9.28]$$

which accounts for simple sentences like *Abigail ate the kimchi* — as we will see, the direct object *the kimchi* is part of the verb phrase. But there are more complex forms of sentences as well:

$$S \rightarrow ADVP NP VP \quad \begin{matrix} Unfortunately Abigail ate the kimchi. \\ Abigail ate the kimchi and Max had a burger. \end{matrix} \quad [9.29]$$

$$S \rightarrow S CC S \quad \begin{matrix} Abigail ate the kimchi and Max had a burger. \\ Eat the kimchi. \end{matrix} \quad [9.30]$$

$$S \rightarrow VP \quad \begin{matrix} Eat the kimchi. \end{matrix} \quad [9.31]$$

- 4787 where ADVP is an adverbial phrase (e.g., *unfortunately*, *very unfortunately*) and CC is a
 4788 coordinating conjunction (e.g., *and*, *but*).⁸

4789 **9.2.3.2 Noun phrases**

Noun phrases refer to entities, real or imaginary, physical or abstract: *Asha*, *the steamed dumpling*, *parts and labor*, *nobody*, *the whiteness of the whale*, and *the rise of revolutionary syndicalism in the early twentieth century*. Noun phrase productions include “bare” nouns, which may optionally follow determiners, as well as pronouns:

$$NP \rightarrow NN | NNS | NNP | PRP \quad [9.32]$$

$$NP \rightarrow DET NN | DET NNS | DET NNP \quad [9.33]$$

- 4790 The tags NN, NNS, and NNP refer to singular, plural, and proper nouns; PRP refers to
 4791 personal pronouns, and DET refers to determiners. The grammar also contains terminal
 4792 productions from each of these tags, e.g., $PRP \rightarrow I | you | we | \dots$.

Noun phrases may be modified by adjectival phrases (ADJP; e.g., *the small Russian dog*) and numbers (CD; e.g., *the five pastries*), each of which may optionally follow a determiner:

$$NP \rightarrow ADJP NN | ADJP NNS | DET ADJP NN | DET ADJP NNS \quad [9.34]$$

$$NP \rightarrow CD NNS | DET CD NNS | \dots \quad [9.35]$$

Some noun phrases include multiple nouns, such as *the liberation movement* and *an antelope horn*, necessitating additional productions:

$$NP \rightarrow NN NN | NN NNS | DET NN NN | \dots \quad [9.36]$$

⁸Notice that the grammar does not include the recursive production $S \rightarrow ADVP S$. It may be helpful to think about why this production would cause the grammar to overgenerate.

4793 These multiple noun constructions can be combined with adjectival phrases and cardinal
 4794 numbers, leading to a large number of additional productions.

Recursive noun phrase productions include coordination, prepositional phrase attachment, subordinate clauses, and verb phrase adjuncts:

$NP \rightarrow NP\ Cc\ NP$	<i>e.g., the red and the black</i>	[9.37]
$NP \rightarrow NP\ PP$	<i>e.g., the President of the Georgia Institute of Technology</i>	[9.38]
$NP \rightarrow NP\ SBAR$	<i>e.g., a whale which he had wounded</i>	[9.39]
$NP \rightarrow NP\ VP$	<i>e.g., a whale taken near Shetland</i>	[9.40]

4795 These recursive productions are a major source of ambiguity, because the VP and PP non-
 4796 terminals can also generate NP children. Thus, the *the President of the Georgia Institute of*
 4797 *Technology* can be derived in two ways, as can *a whale taken near Shetland in October*.

4798 But aside from these few recursive productions, the noun phrase fragment of the Penn
 4799 Treebank grammar is relatively flat, containing a large of number of productions that go
 4800 from NP directly to a sequence of parts-of-speech. If noun phrases had more internal
 4801 structure, the grammar would need fewer rules, which, as we will see, would make pars-
 4802 ing faster and machine learning easier. Vadas and Curran (2011) propose to add additional
 4803 structure in the form of a new non-terminal called a **nominal modifier** (NML), e.g.,

4804 (9.17) (NP (NN crude) (NN oil) (NNS prices)) (PTB analysis)
 4805 (NP (NML (NN crude) (NN oil)) (NNS prices)) (NML-style analysis)

4806 Another proposal is to treat the determiner as the head of a **determiner phrase** (DP;
 4807 Abney, 1987). There are linguistic arguments for and against determiner phrases (e.g.,
 4808 Van Eynde, 2006). From the perspective of context-free grammar, DPs enable more struc-
 4809 tured analyses of some constituents, e.g.,

4810 (9.18) (NP (DT the) (JJ white) (NN whale)) (PTB analysis)
 4811 (DP (DT the) (NP (JJ white) (NN whale))) (DP-style analysis).

4812 9.2.3.3 Verb phrases

Verb phrases describe actions, events, and states of being. The PTB tagset distinguishes several classes of verb inflections: base form (VB; *she likes to snack*), present-tense third-person singular (VBD; *she snacks*), present tense but not third-person singular (VBP; *they snack*), past tense (VBD; *they snacked*), present participle (VBG; *they are snacking*), and past participle (VBN; *they had snacked*).⁹ Each of these forms can constitute a verb phrase on its

⁹It bears emphasis the principles governing this tagset design are entirely English-specific: VBP is a meaningful category only because English morphology distinguishes third-person singular from all person-number combinations.

own:

$$VP \rightarrow VB \mid VBZ \mid VBD \mid VBN \mid VBG \mid VBP \quad [9.41]$$

More complex verb phrases can be formed by a number of recursive productions, including the use of coordination, modal verbs (MD; *she should snack*), and the infinitival *to* (TO):

$VP \rightarrow MD \ VP$	<i>She will snack</i>	[9.42]
$VP \rightarrow VBD \ VP$	<i>She had snacked</i>	[9.43]
$VP \rightarrow VBZ \ VP$	<i>She has been snacking</i>	[9.44]
$VP \rightarrow VBN \ VP$	<i>She has been snacking</i>	[9.45]
$VP \rightarrow TO \ VP$	<i>She wants to snack</i>	[9.46]
$VP \rightarrow VP \ CC \ VP$	<i>She buys and eats many snacks</i>	[9.47]

- 4813 Each of these productions uses recursion, with the VP non-terminal appearing in both the
 4814 LHS and RHS. This enables the creation of complex verb phrases, such as *She will have*
 4815 *wanted to have been snacking*.

Transitive verbs take noun phrases as direct objects, and ditransitive verbs take two direct objects:

$VP \rightarrow VBZ \ NP$	<i>She teaches algebra</i>	[9.48]
$VP \rightarrow VBG \ NP$	<i>She has been teaching algebra</i>	[9.49]
$VP \rightarrow VBD \ NP \ NP$	<i>She taught her brother algebra</i>	[9.50]

These productions are *not* recursive, so a unique production is required for each verb part-of-speech. They also do not distinguish transitive from intransitive verbs, so the resulting grammar overgenerates examples like **She sleeps sushi* and **She learns Boyang algebra*. Sentences can also be direct objects:

$VP \rightarrow VBZ \ S$	<i>Asha wants to eat the kimchi</i>	[9.51]
$VP \rightarrow VBZ \ SBAR$	<i>Asha knows that Boyang eats the kimchi</i>	[9.52]

- 4816 The first production overgenerates, licensing sentences like **Asha sees Boyang eats the kim-*
 4817 *chi*. This problem could be addressed by designing a more specific set of sentence non-
 4818 terminals, indicating whether the main verb can be conjugated.

Verbs can also be modified by prepositional phrases and adverbial phrases:

$VP \rightarrow VBZ \ PP$	<i>She studies at night</i>	[9.53]
$VP \rightarrow VBZ \ ADVP$	<i>She studies intensively</i>	[9.54]
$VP \rightarrow ADVP \ VBG$	<i>She is not studying</i>	[9.55]

4819 Again, because these productions are not recursive, the grammar must include produc-
 4820 tions for every verb part-of-speech.

A special set of verbs, known as **copula**, can take **predicative adjectives** as direct ob-
 jects:

$VP \rightarrow VBZ\ ADJP$	<i>She is hungry</i>	[9.56]
$VP \rightarrow VBP\ ADJP$	<i>Success seems increasingly unlikely</i>	[9.57]

4821 The PTB does not have a special non-terminal for copular verbs, so this production gen-
 4822 erates non-grammatical examples such as **She eats tall*.

Particles (PRT as a phrase; RP as a part-of-speech) work to create phrasal verbs:

$VP \rightarrow VB\ PRT$	<i>She told them to fuck off</i>	[9.58]
$VP \rightarrow VBD\ PRT\ NP$	<i>They gave up their ill-gotten gains</i>	[9.59]

4823 As the second production shows, particle productions are required for all configura-
 4824 tions of verb parts-of-speech and direct objects.

4825 9.2.3.4 Other constituents

The remaining constituents require far fewer productions. **Prepositional phrases** almost always consist of a preposition and a noun phrase,

$PP \rightarrow IN\ NP$	<i>the whiteness of the whale</i>	[9.60]
$PP \rightarrow TO\ NP$	<i>What the white whale was to Ahab, has been hinted.</i>	[9.61]

Similarly, complement clauses consist of a complementizer (usually a preposition, possibly null) and a sentence,

$SBAR \rightarrow IN\ S$	<i>She said that it was spicy</i>	[9.62]
$SBAR \rightarrow S$	<i>She said it was spicy</i>	[9.63]

Adverbial phrases are usually bare adverbs ($ADVP \rightarrow RB$), with a few exceptions:

$ADVP \rightarrow RB\ RBR$	<i>They went considerably further</i>	[9.64]
$ADVP \rightarrow ADVP\ PP$	<i>They went considerably further than before</i>	[9.65]

4826 The tag RBR is a comparative adverb.

Adjectival phrases extend beyond bare adjectives ($\text{ADJP} \rightarrow \text{JJ}$) in a number of ways:

$\text{ADJP} \rightarrow \text{RB JJ}$	<i>very hungry</i>	[9.66]
$\text{ADJP} \rightarrow \text{RBR JJ}$	<i>more hungry</i>	[9.67]
$\text{ADJP} \rightarrow \text{JJS JJ}$	<i>best possible</i>	[9.68]
$\text{ADJP} \rightarrow \text{RB JJR}$	<i>even bigger</i>	[9.69]
$\text{ADJP} \rightarrow \text{JJ CC JJ}$	<i>high and mighty</i>	[9.70]
$\text{ADJP} \rightarrow \text{JJ JJ}$	<i>West German</i>	[9.71]
$\text{ADJP} \rightarrow \text{RB VBN}$	<i>previously reported</i>	[9.72]

4827 The tags JJR and JJS refer to comparative and superlative adjectives respectively.

All of these phrase types can be coordinated:

$\text{PP} \rightarrow \text{PP CC PP}$	<i>on time and under budget</i>	[9.73]
$\text{ADVP} \rightarrow \text{ADVP CC ADVP}$	<i>now and two years ago</i>	[9.74]
$\text{ADJP} \rightarrow \text{ADJP CC ADJP}$	<i>quaint and rather deceptive</i>	[9.75]
$\text{SBar} \rightarrow \text{SBar CC SBar}$	<i>whether they want control</i>	[9.76]
	<i>or whether they want exports</i>	

4828 9.2.4 Grammatical ambiguity

4829 Context-free parsing is useful not only because it determines whether a sentence is grammatical, but mainly because the constituents and their relations can be applied to tasks 4830 such as information extraction (chapter 17) and sentence compression (Jing, 2000; Clarke 4831 and Lapata, 2008). However, the **ambiguity** of wide-coverage natural language grammars 4832 poses a serious problem for such potential applications. As an example, Figure 9.13 shows 4833 two possible analyses for the simple sentence *We eat sushi with chopsticks*, depending on 4834 whether the *chopsticks* modify *eat* or *sushi*. Realistic grammars can license thousands or 4835 even millions of parses for individual sentences. **Weighted context-free grammars** solve 4836 this problem by attaching weights to each production, and selecting the derivation with 4837 the highest score. This is the focus of chapter 10. 4838

4839 9.3 *Mildly context-sensitive languages

4840 Beyond context-free languages lie **context-sensitive languages**, in which the expansion 4841 of a non-terminal depends on its neighbors. In the general class of context-sensitive 4842 languages, computation becomes much more challenging: the membership problem for 4843 context-sensitive languages is PSPACE-complete. Since PSPACE contains the complexity 4844 class NP (problems that can be solved in polynomial time on a non-deterministic Turing

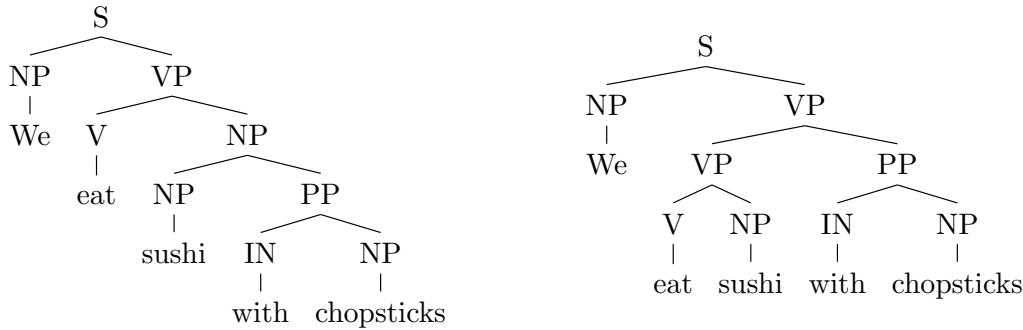


Figure 9.13: Two derivations of the same sentence

4845 machine), PSPACE-complete problems cannot be solved efficiently if $P \neq NP$. Thus, de-
 4846 signing an efficient parsing algorithm for the full class of context-sensitive languages is
 4847 probably hopeless.¹⁰

4848 However, Joshi (1985) identifies a set of properties that define **mildly context-sensitive**
 4849 **languages**, which are a strict subset of context-sensitive languages. Like context-free lan-
 4850 guages, mildly context-sensitive languages are efficiently parseable. However, the mildly
 4851 context-sensitive languages include non-context-free languages, such as the “copy lan-
 4852 guage” $\{ww \mid w \in \Sigma^*\}$ and the language $a^m b^n c^m d^n$. Both are characterized by **cross-**
 4853 **serial dependencies**, linking symbols at long distance across the string.¹¹ For example, in
 4854 the language $a^n b^m c^n d^m$, each a symbol is linked to exactly one c symbol, regardless of the
 4855 number of intervening b symbols.

4856 9.3.1 Context-sensitive phenomena in natural language

4857 Such phenomena are occasionally relevant to natural language. A classic example is found
 4858 in Swiss-German (Shieber, 1985), in which sentences such as *we let the children help Hans*
 4859 *paint the house* are realized by listing all nouns before all verbs, i.e., *we the children Hans the*
 4860 *house let help paint*. Furthermore, each noun’s determiner is dictated by the noun’s **case**
 4861 **marking** (the role it plays with respect to the verb). Using an argument that is analogous
 4862 to the earlier discussion of center-embedding (§ 9.2), Shieber argues that these case mark-
 4863 ing constraints are a cross-serial dependency, homomorphic to $a^m b^n c^m d^n$, and therefore
 4864 not context-free.

¹⁰If $P \neq NP$, then it contains problems that cannot be solved in polynomial time on a non-deterministic Turing machine; equivalently, solutions to these problems cannot even be checked in polynomial time (Arora and Barak, 2009).

¹¹A further condition of the set of mildly-context-sensitive languages is *constant growth*: if the strings in the language are arranged by length, the gap in length between any pair of adjacent strings is bounded by some language specific constant. This condition excludes languages such as $\{a^{2^n} \mid n \geq 0\}$.

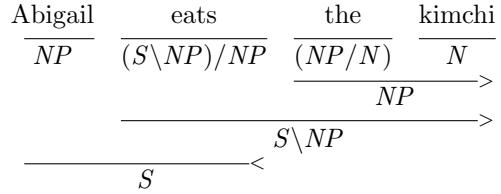


Figure 9.14: A syntactic analysis in CCG involving forward and backward function application

As with the move from regular to context-free languages, mildly context-sensitive languages can be motivated by expedience. While infinite sequences of cross-serial dependencies cannot be handled by context-free grammars, even finite sequences of cross-serial dependencies are more convenient to handle using a mildly context-sensitive formalism like **tree-adjoining grammar** (TAG) and **combinatory categorial grammar** (CCG). Furthermore, TAG-inspired parsers have been shown to be particularly effective in parsing the Penn Treebank (Collins, 1997; Carreras et al., 2008), and CCG plays a leading role in current research on semantic parsing (Zettlemoyer and Collins, 2005). Furthermore, these two formalisms are weakly equivalent: any language that can be specified in TAG can also be specified in CCG, and vice versa (Joshi et al., 1991). The remainder of the chapter gives a brief overview of CCG, but you are encouraged to consult Joshi and Schabes (1997) and Steedman and Baldridge (2011) for more detail on TAG and CCG respectively.

9.3.2 Combinatory categorial grammar

In combinatory categorial grammar, structural analyses are built up through a small set of generic combinatorial operations, which apply to immediately adjacent sub-structures. These operations act on the categories of the sub-structures, producing a new structure with a new category. The basic categories include S (sentence), NP (noun phrase), VP (verb phrase) and N (noun). The goal is to label the entire span of text as a sentence, S.

Complex categories, or types, are constructed from the basic categories, parentheses, and forward and backward slashes: for example, S/NP is a complex type, indicating a sentence that is lacking a noun phrase to its right; $S\backslash NP$ is a sentence lacking a noun phrase to its left. Complex types act as functions, and the most basic combinatory operations are function application to either the right or left neighbor. For example, the type of a verb phrase, such as *eats*, would be $S\backslash NP$. Applying this function to a subject noun phrase to its left results in an analysis of *Abigail eats* as category S, indicating a successful parse.

Transitive verbs must first be applied to the direct object, which in English appears to the right of the verb, before the subject, which appears on the left. They therefore have the more complex type $(S\backslash NP)/NP$. Similarly, the application of a determiner to the noun at

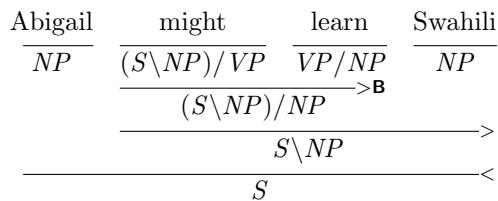


Figure 9.15: A syntactic analysis in CCG involving function composition (example modified from Steedman and Baldridge, 2011)

its right results in a noun phrase, so determiners have the type NP/N. Figure 9.14 provides an example involving a transitive verb and a determiner. A key point from this example is that it can be trivially transformed into phrase-structure tree, by treating each function application as a constituent phrase. Indeed, when CCG's only combinatory operators are forward and backward function application, it is equivalent to context-free grammar. However, the location of the "effort" has changed. Rather than designing good productions, the grammar designer must focus on the **lexicon** — choosing the right categories for each word. This makes it possible to parse a wide range of sentences using only a few generic combinatory operators.

Things become more interesting with the introduction of two additional operators: **composition** and **type-raising**. Function composition enables the combination of complex types: $X/Y \circ Y/Z \Rightarrow_B X/Z$ (forward composition) and $Y\backslash Z \circ X\backslash Y \Rightarrow_B X\backslash Z$ (backward composition).¹² Composition makes it possible to “look inside” complex types, and combine two adjacent units if the “input” for one is the “output” for the other. Figure 9.15 shows how function composition can be used to handle modal verbs. While this sentence can be parsed using only function application, the composition-based analysis is preferable because the unit *might learn* functions just like a transitive verb, as in the example *Abigail studies Swahili*. This in turn makes it possible to analyze conjunctions such as *Abigail studies and might learn Swahili*, attaching the direct object *Swahili* to the entire conjoined verb phrase *studies and might learn*. The Penn Treebank grammar fragment from § 9.2.3 would be unable to handle this case correctly: the direct object *Swahili* could attach only to the second verb *learn*.

4916 Type raising converts an element of type X to a more complex type: $X \Rightarrow_T T/(T \setminus X)$
4917 (forward type-raising to type T), and $X \Rightarrow_T T \setminus (T/X)$ (backward type-raising to type
4918 T). Type-raising makes it possible to reverse the relationship between a function and its
4919 argument — by transforming the argument into a function over functions over arguments!
4920 An example may help. Figure 9.15 shows how to analyze an object relative clause, *a story*
4921 *that Abigail tells*. The problem is that *tells* is a transitive verb, expecting a direct object to
4922 its right. As a result, *Abigail tells* is not a valid constituent. The issue is resolved by raising

¹²The subscript B follows notation from Curry and Feys (1958).

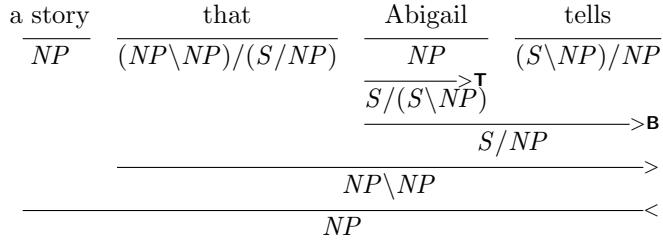


Figure 9.16: A syntactic analysis in CCG involving an object relative clause (based on slides from Alex Clark)

4923 *Abigail* from NP to the complex type $(S / NP) \setminus NP$. This function can then be combined
 4924 with the transitive verb *tells* by forward composition, resulting in the type (S / NP) , which
 4925 is a sentence lacking a direct object to its right.¹³ From here, we need only design the
 4926 lexical entry for the complementizer *that* to expect a right neighbor of type (S / NP) , and
 4927 the remainder of the derivation can proceed by function application.

4928 Composition and type-raising give CCG considerable power and flexibility, but at a
 4929 price. The simple sentence *Abigail tells Max* can be parsed in two different ways: by func-
 4930 tion application (first forming the verb phrase *tells Max*), and by type-raising and compo-
 4931 sition (first forming the non-constituent *Abigail tells*). This **derivational ambiguity** does
 4932 not affect the resulting linguistic analysis, so it is sometimes known as **spurious ambi-**
 4933 **guity**. Hockenmaier and Steedman (2007) present a translation algorithm for converting
 4934 the Penn Treebank into CCG derivations, using composition and type-raising only when
 4935 necessary.

4936 Exercises

- 4937 1. Sketch out the state diagram for finite-state acceptors for the following languages
 4938 on the alphabet $\{a, b\}$.
- 4939 a) Even-length strings. (Be sure to include 0 as an even number.)
- 4940 b) Strings that contain *aaa* as a substring.
- 4941 c) Strings containing an even number of *a* and an odd number of *b* symbols.
- 4942 d) Strings in which the substring *bbb* must be terminal if it appears — the string
 4943 need not contain *bbb*, but if it does, nothing can come after it.
- 4944 2. Levenshtein edit distance is the number of insertions, substitutions, or deletions
 4945 required to convert one string to another.

¹³The missing direct object would be analyzed as a **trace** in CFG-like approaches to syntax, including the Penn Treebank.

- 4946 a) Define a finite-state acceptor that accepts all strings with edit distance 1 from
 4947 the target string, *target*.
 4948 b) Now think about how to generalize your design to accept all strings with edit
 4949 distance from the target string equal to d . If the target string has length ℓ , what
 4950 is the minimal number of states required?
- 4951 3. Construct an FSA in the style of Figure 9.3, which handles the following examples:

- 4952 • *nation*/N, *national*/ADJ, *nationalize*/V, *nationalizer*/N
 4953 • *America*/N, *American*/ADJ, *Americanize*/V, *Americanizer*/N

4954 Be sure that your FSA does not accept any further derivations, such as **nationalizeral*
 4955 and **Americanizern*.

- 4956 4. Show how to construct a trigram language model in a weighted finite-state acceptor.
 4957 Make sure that you handle the edge cases at the beginning and end of the sequence
 4958 accurately.
- 4959 5. Extend the FST in Figure 9.6 to handle the other two parts of rule 1a of the Porter
 4960 stemmer: *-sses* → *ss*, and *-ies* → *-i*.

- 4961 6. § 9.1.4.4 describes T_O , a transducer that captures English orthography by transduc-
 4962 ing *cook + ed* → *cooked* and *bake + ed* → *baked*. Design an unweighted finite-state
 4963 transducer that captures this property of English orthography.

4964 Next, augment the transducer to appropriately model the suffix *-s* when applied to
 4965 words ending in *s*, e.g. *kiss+s* → *kisses*.

- 4966 7. Add parenthesization to the grammar in Figure 9.11 so that it is no longer ambigu-
 4967 ous.
- 4968 8. Construct three examples — a noun phrase, a verb phrase, and a sentence — which
 4969 can be derived from the Penn Treebank grammar fragment in § 9.2.3, yet are not
 4970 grammatical. Avoid reusing examples from the text. Optionally, propose corrections
 4971 to the grammar to avoid generating these cases.
- 4972 9. Produce parses for the following sentences, using the Penn Treebank grammar frag-
 4973 ment from § 9.2.3.

- 4974 (9.19) This aggression will not stand.
 4975 (9.20) I can get you a toe.
 4976 (9.21) Sometimes you eat the bar and sometimes the bar eats you.

4977 Then produce parses for three short sentences from a news article from this week.

4978 10. * One advantage of CCG is its flexibility in handling coordination:

4979 (9.22) *Abigail and Max speak Swahili*

4980 (9.23) *Abigail speaks and Max understands Swahili*

Define the lexical entry for *and* as

$$\textit{and} := (X/X) \setminus X, \quad [9.77]$$

4981 where X can refer to any type. Using this lexical entry, show how to parse the two
4982 examples above. In the second example, *Swahili* should be combined with the coor-
4983 dination *Abigail speaks and Max understands*, and not just with the verb *understands*.

4984 **Chapter 10**

4985 **Context-free parsing**

4986 Parsing is the task of determining whether a string can be derived from a given context-
4987 free grammar, and if so, how. The parse structure can answer basic questions of who-did-
4988 what-to-whom, and is useful for various downstream tasks, such as semantic analysis
4989 (chapter 12 and 13) and information extraction (chapter 17).

For a given input and grammar, how many parse trees are there? Consider a minimal context-free grammar with only one non-terminal, X , and the following productions:

$$\begin{aligned} X \rightarrow & X \ X \\ X \rightarrow & aardvark \mid abacus \mid \dots \mid zyther \end{aligned}$$

The second line indicates unary productions to every nonterminal in Σ . In this grammar, the number of possible derivations for a string w is equal to the number of binary bracketings, e.g.,

$$(((w_1 w_2) w_3) w_4) w_5), \quad (((w_1 (w_2 w_3)) w_4) w_5), \quad ((w_1 (w_2 (w_3 w_4))) w_5), \quad \dots$$

4990 The number of such bracketings is a **Catalan number**, which grows super-exponentially
4991 in the length of the sentence, $C_n = \frac{(2n)!}{(n+1)n!}$. As with sequence labeling, it is only possible to
4992 exhaustively search the space of parses by resorting to locality assumptions, which make it
4993 possible to search efficiently by reusing shared substructures with dynamic programming.
4994 This chapter focuses on a bottom-up dynamic programming algorithm, which enables
4995 exhaustive search of the space of possible parses, but imposes strict limitations on the
4996 form of scoring function. These limitations can be relaxed by abandoning exhaustive
4997 search. Non-exact search methods will be briefly discussed at the end of this chapter, and
4998 one of them — **transition-based parsing** — will be the focus of chapter 11.

S	\rightarrow	NP VP
NP	\rightarrow	NP PP <i>we</i> <i>sushi</i> <i>chopsticks</i>
PP	\rightarrow	IN NP
IN	\rightarrow	<i>with</i>
VP	\rightarrow	V NP VP PP
V	\rightarrow	<i>eat</i>

Table 10.1: A toy example context-free grammar

4999 10.1 Deterministic bottom-up parsing

5000 The **CKY algorithm**¹ is a bottom-up approach to parsing in a context-free grammar. It
 5001 efficiently tests whether a string is in a language, without enumerating all possible parses.
 5002 The algorithm first forms small constituents, and then tries to merge them into larger
 5003 constituents.

5004 To understand the algorithm, consider the input, *We eat sushi with chopsticks*. According-
 5005 ing to the toy grammar in Table 10.1, each terminal symbol can be generated by exactly
 5006 one unary production, resulting in the sequence NP V NP IN NP. Next, we try to apply
 5007 binary productions to merge adjacent symbols into larger constituents: for example, V
 5008 NP can be merged into a verb phrase (VP), and IN NP can be merged into a prepositional
 5009 phrase (PP). Bottom-up parsing tries to find some series of mergers that ultimately results
 5010 in the start symbol S covering the entire input.

5011 The CKY algorithm systematizes this approach, incrementally constructing a table t in
 5012 which each cell $t[i, j]$ contains the set of nonterminals that can derive the span $w_{i+1:j}$. The
 5013 algorithm fills in the upper right triangle of the table; it begins with the diagonal, which
 5014 corresponds to substrings of length 1, and then computes derivations for progressively
 5015 larger substrings, until reaching the upper right corner $t[0, M]$, which corresponds to the
 5016 entire input, $w_{1:M}$. If the start symbol S is in $t[0, M]$, then the string w is in the language
 5017 defined by the grammar. This process is detailed in Algorithm 13, and the resulting data
 5018 structure is shown in Figure 10.1. Informally, here's how it works:

- 5019 • Begin by filling in the diagonal: the cells $t[m - 1, m]$ for all $m \in \{1, 2, \dots, M\}$. These
 5020 cells are filled with terminal productions that yield the individual tokens; for the
 5021 word $w_2 = \text{sushi}$, we fill in $t[1, 2] = \{\text{NP}\}$, and so on.
- 5022 • Then fill in the next diagonal, in which each cell corresponds to a subsequence of
 5023 length two: $t[0, 2], t[1, 3], \dots, t[M - 2, M]$. These cells are filled in by looking for
 5024 binary productions capable of producing at least one entry in each of the cells corre-

¹The name is for Cocke-Kasami-Younger, the inventors of the algorithm. It is a special case **chart parsing**, because its stores reusable computations in a chart-like data structure.

Algorithm 13 The CKY algorithm for parsing a sequence $w \in \Sigma^*$ in a context-free grammar $G = (N, \Sigma, R, S)$, with non-terminals N , production rules R , and start symbol S . The grammar is assumed to be in Chomsky normal form (§ 9.2.1.2). The function $\text{PICKFROM}(b[i, j, X])$ selects an element of the set $b[i, j, X]$ arbitrarily. All values of t and b are initialized to \emptyset .

```

1: procedure CKY( $w, G = (N, \Sigma, R, S)$ )
2:   for  $m \in \{1 \dots M\}$  do
3:      $t[m - 1, m] \leftarrow \{X : (X \rightarrow w_m) \in R\}$ 
4:   for  $\ell \in \{2, 3, \dots, M\}$  do                                 $\triangleright$  Iterate over constituent lengths
5:     for  $m \in \{0, 1, \dots, M - \ell\}$  do           $\triangleright$  Iterate over left endpoints
6:       for  $k \in \{m + 1, m + 2, \dots, m + \ell - 1\}$  do       $\triangleright$  Iterate over split points
7:         for  $(X \rightarrow Y Z) \in R$  do           $\triangleright$  Iterate over rules
8:           if  $Y \in t[m, k] \wedge Z \in t[k, m + \ell]$  then
9:              $t[m, m + \ell] \leftarrow t[m, m + \ell] \cup X$            $\triangleright$  Add non-terminal to table
10:             $b[m, m + \ell, X] \leftarrow b[m, m + \ell, X] \cup (Y, Z, k)$        $\triangleright$  Add back-pointers
11:   if  $S \in t[0, M]$  then
12:     return TRACEBACK( $S, 0, M, b$ )
13:   else
14:     return  $\emptyset$ 
15: procedure TRACEBACK( $X, i, j, b$ )
16:   if  $j = i + 1$  then
17:     return  $X$ 
18:   else
19:      $(Y, Z, k) \leftarrow \text{PICKFROM}(b[i, j, X])$ 
20:     return  $X \rightarrow (\text{TRACEBACK}(Y, i, k, b), \text{TRACEBACK}(Z, k, j, b))$ 

```

5025 sponding to left and right children. For example, the cell $t[1, 3]$ includes VP because
 5026 the grammar includes the production $\text{VP} \rightarrow \text{V NP}$, and the chart contains $\text{V} \in t[1, 2]$
 5027 and $\text{NP} \in t[2, 3]$.

- 5028 • At the next diagonal, the entries correspond to spans of length three. At this level,
 5029 there is an additional decision at each cell: where to split the left and right children.
 5030 The cell $t[i, j]$ corresponds to the subsequence $w_{i+1:j}$, and we must choose some
 5031 *split point* $i < k < j$, so that $w_{i+1:k}$ is the left child and $w_{k+1:j}$ is the right child. We
 5032 consider all possible k , looking for productions that generate elements in $t[i, k]$ and
 5033 $t[k, j]$; the left-hand side of all such productions can be added to $t[i, j]$. When it is
 5034 time to compute $t[i, j]$, the cells $t[i, k]$ and $t[k, j]$ are guaranteed to be complete, since
 5035 these cells correspond to shorter sub-strings of the input.
- 5036 • The process continues until we reach $t[0, M]$.

5037 Figure 10.1 shows the chart that arises from parsing the sentence *We eat sushi with chop-*
 5038 *sticks* using the grammar defined above.

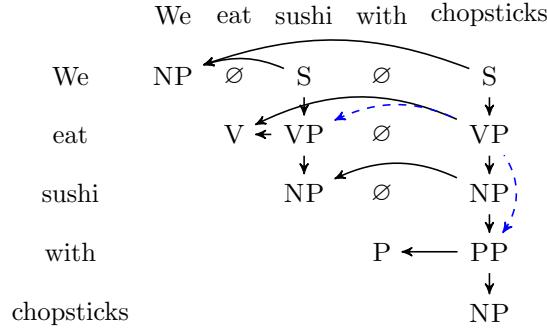


Figure 10.1: An example completed CKY chart. The solid and dashed lines show the back pointers resulting from the two different derivations of VP in position $t[1, 5]$.

5039 10.1.1 Recovering the parse tree

5040 As with the Viterbi algorithm, it is possible to identify a successful parse by storing and
 5041 traversing an additional table of back-pointers. If we add an entry X to cell $t[i, j]$ by using
 5042 the production $X \rightarrow YZ$ and the split point k , then we store the back-pointer $b[i, j, X] =$
 5043 (Y, Z, k) . Once the table is complete, we can recover a parse by tracing this pointers,
 5044 starting at $b[0, M, S]$, and stopping when they ground out at terminal productions.

5045 For ambiguous sentences, there will be multiple paths to reach $S \in t[0, M]$. For exam-
 5046 ple, in Figure 10.1, the goal state $S \in t[0, M]$ is reached through the state $VP \in t[1, 5]$, and
 5047 there are two different ways to generate this constituent: one with *(eat sushi)* and *(with
 5048 chopsticks)* as children, and another with *(eat)* and *(sushi with chopsticks)* as children. The
 5049 presence of multiple paths indicates that the input can be generated by the grammar in
 5050 more than one way. In Algorithm 13, one of these derivations is selected arbitrarily. As
 5051 discussed in § 10.3, **weighted context-free grammars** can select a single parse that maxi-
 5052 mizes a scoring function.

5053 10.1.2 Non-binary productions

5054 The CKY algorithm assumes that all productions with non-terminals on the right-hand
 5055 side (RHS) are binary. But in real grammars, such as the one considered in chapter 9,
 5056 there will be productions with more than two elements on the right-hand side, and other
 5057 productions with only a single element.

- 5058 • Productions with more than two elements on the right-hand side can be **binarized**
 5059 by creating additional non-terminals, as described in § 9.2.1.2. For example, given
 5060 the production $VP \rightarrow V NP NP$ (for ditransitive verbs), we can convert to $VP \rightarrow$
 5061 $VP_{ditrans}/NP NP$, and then add the production $VP_{ditrans}/NP \rightarrow V NP$.

- What about unary productions like $VP \rightarrow V$? In practice, this is handled by making a second pass on each diagonal, in which each cell $t[i, j]$ is augmented with all possible unary productions capable of generating each item already in the cell — formally, $t[i, j]$ is extended to its **unary closure**. Suppose the example grammar in Table 10.1 were extended to include the production $VP \rightarrow V$, enabling sentences with intransitive verb phrases, like *we eat*. Then the cell $t[1, 2]$ — corresponding to the word *eat* — would first include the set $\{V\}$, and would be augmented to the set $\{V, VP\}$ during this second pass.

10.1.3 Complexity

For an input of length M and a grammar with R productions and N non-terminals, the space complexity of the CKY algorithm is $\mathcal{O}(M^2N)$: the number of cells in the chart is $\mathcal{O}(M^2)$, and each cell must hold $\mathcal{O}(N)$ elements. The time complexity is $\mathcal{O}(M^3R)$: each cell is computed by searching over $\mathcal{O}(M)$ split points, with R possible productions for each split point. Both the time and space complexity are considerably worse than the Viterbi algorithm, which is linear in the length of the input.

10.2 Ambiguity

Syntactic ambiguity is endemic to natural language. Here are a few broad categories:

- **Attachment ambiguity:** e.g., *We eat sushi with chopsticks, I shot an elephant in my pajamas*. In these examples, the prepositions (*with, in*) can attach to either the verb or the direct object.
- **Modifier scope:** e.g., *southern food store, plastic cup holder*. In these examples, the first word could be modifying the subsequent adjective, or the final noun.
- **Particle versus preposition:** e.g., *The puppy tore up the staircase*. Phrasal verbs like *tore up* often include particles which could also act as prepositions. This has structural implications: if *up* is a preposition, then *up the staircase* is a prepositional phrase; if *up* is a particle, then *the staircase* is the direct object to the verb.
- **Complement structure:** e.g., *The students complained to the professor that they didn't understand*. This is another form of attachment ambiguity, where the complement *that they didn't understand* could attach to the main verb (*complained*), or to the indirect object (*the professor*).
- **Coordination scope:** e.g., *"I see," said the blind man, as he picked up the hammer and saw*. In this example, the lexical ambiguity for *saw* enables it to be coordinated either with the noun *hammer* or the verb *picked up*.

These forms of ambiguity can combine, so that seemingly simple headlines like *Fed raises interest rates* have dozens of possible analyses even in a minimal grammar. In a broad coverage grammar, typical sentences can have millions of parses. While careful grammar design can chip away at this ambiguity, a better strategy is to combine broad coverage parsers with data driven strategies for identifying the correct analysis.

10.2.1 Parser evaluation

Before continuing to parsing algorithms that are able to handle ambiguity, we stop to consider how to measure parsing performance. Suppose we have a set of *reference parses* — the ground truth — and a set of *system parses* that we would like to score. A simple solution would be per-sentence accuracy: the parser is scored by the proportion of sentences on which the system and reference parses exactly match.² But as any good student knows, it is better to get *partial credit*, which we can assign to analyses that correctly match parts of the reference parse. The PARSEval metrics (Grishman et al., 1992) score each system parse via:

Precision: the fraction of constituents in the system parse that match a constituent in the reference parse.

Recall: the fraction of constituents in the reference parse that match a constituent in the system parse.

In **labeled precision** and **recall**, the system must also match the phrase type for each constituent; in **unlabeled precision** and **recall**, it is only required to match the constituent structure. As in chapter 4, the precision and recall can be combined into an *F*-MEASURE, $F = \frac{2 \times P \times R}{P + R}$.

In Figure 10.2, suppose that the left tree is the system parse and the right tree is the reference parse. We have the following spans:

- $S \rightarrow w_{1:5}$ is *true positive*, because it appears in both trees.
- $VP \rightarrow w_{2:5}$ is *true positive* as well.
- $NP \rightarrow w_{3:5}$ is *false positive*, because it appears only in the system output.
- $PP \rightarrow w_{4:5}$ is *true positive*, because it appears in both trees.
- $VP \rightarrow w_{2:3}$ is *false negative*, because it appears only in the reference.

²Most parsing papers do not report results on this metric, but Finkel et al. (2008) find that a strong parser finds the exact correct parse on 35% of sentences of length ≤ 40 , and on 62% of parses of length ≤ 15 in the Penn Treebank.

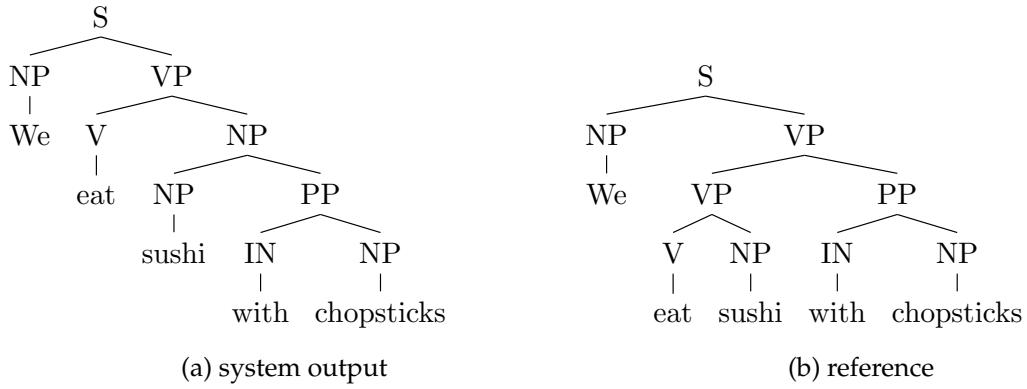


Figure 10.2: Two possible analyses from the grammar in Table 10.1

5124 The labeled and unlabeled precision of this parse is $\frac{3}{4} = 0.75$, and the recall is $\frac{3}{4} = 0.75$, for
 5125 an F-measure of 0.75. For an example in which precision and recall are not equal, suppose
 5126 the reference parse instead included the production $VP \rightarrow V NP PP$. In this parse, the
 5127 reference does not contain the constituent $w_{2:3}$, so the recall would be 1.³

5128 10.2.2 Local solutions

5129 Some ambiguity can be resolved locally. Consider the following examples,

5130 (10.1) We met the President on Monday.

5131 (10.2) We met the President of Mexico.

Each case ends with a preposition, which can be attached to the verb *met* or the noun phrase *the president*. This ambiguity can be resolved by using a labeled corpus to compare the likelihood of observing the preposition alongside each candidate attachment point,

$$p(on \mid met) \geq p(on \mid President) \quad [10.1]$$

$$p(of \mid met) \geq p(of \mid President). \quad [10.2]$$

5132 A comparison of these probabilities would successfully resolve this case (Hindle and
5133 Rooth, 1993). Other cases, such as the example ... *eat sushi with chopsticks*, require consider-
5134 ing the object of the preposition — consider the alternative ... *eat sushi with soy sauce*. With
5135 sufficient labeled data, the problem of prepositional phrase attachment can be treated as
5136 a classification task (Ratnaparkhi et al., 1994).

³While the grammar must be binarized before applying the CKY algorithm, evaluation is performed on the original parses. It is therefore necessary to “unbinarize” the output of a CKY-based parser, converting it back to the original grammar.

5137 However, there are inherent limitations to local solutions. While toy examples may
 5138 have just a few ambiguities to resolve, realistic sentences have thousands or millions of
 5139 possible parses. Furthermore, attachment decisions are interdependent, as shown in the
 5140 garden path example:

5141 (10.3) Cats scratch people with claws with knives.

5142 We may want to attach *with claws* to *scratch*, as would be correct in the shorter sentence
 5143 in *cats scratch people with claws*. But this leaves nowhere to attach *with knives*. The cor-
 5144 rect interpretation can be identified only by considering the attachment decisions jointly.
 5145 The huge number of potential parses may seem to make exhaustive search impossible.
 5146 But as with sequence labeling, locality assumptions make it possible to search this space
 5147 efficiently.

5148 10.3 Weighted Context-Free Grammars

5149 Let us define a derivation τ as a set of **anchored productions**,

$$\tau = \{X \rightarrow \alpha, (i, j, k)\}, \quad [10.3]$$

5150 with X corresponding to the left-hand side non-terminal and α corresponding to the right-
 5151 hand side. For grammars in Chomsky normal form, α is either a pair of non-terminals or
 5152 a terminal symbol. The indices i, j, k anchor the production in the input, with X deriving
 5153 the span $w_{i+1:j}$. For binary productions, $w_{i+1:k}$ indicates the span of the left child, and
 5154 $w_{k+1:j}$ indicates the span of the right child; for unary productions, k is ignored. For an
 5155 input w , the optimal parse is then,

$$\hat{\tau} = \underset{\tau \in \mathcal{T}(w)}{\operatorname{argmax}} \Psi(\tau), \quad [10.4]$$

5156 where $\mathcal{T}(w)$ is the set of derivations that yield the input w .

5157 The scoring function Ψ decomposes across anchored productions,

$$\Psi(\tau) = \sum_{(X \rightarrow \alpha, (i, j, k)) \in \tau} \psi(X \rightarrow \alpha, (i, j, k)). \quad [10.5]$$

5158 This is a locality assumption, akin to the assumption in Viterbi sequence labeling. In this
 5159 case, the assumption states that the overall score is a sum over scores of productions,
 5160 which are computed independently. In a **weighted context-free grammar** (WCFG), the
 5161 score of each anchored production $X \rightarrow (\alpha, i, j, k)$ is simply $\psi(X \rightarrow \alpha)$, ignoring the
 5162 anchors (i, j, k) . In other parsing models, the anchors can be used to access features of the
 5163 input, while still permitting efficient bottom-up parsing.

		$\psi(\cdot)$	$\exp \psi(\cdot)$
S	$\rightarrow \text{NP VP}$	0	1
NP	$\rightarrow \text{NP PP}$	-1	$\frac{1}{2}$
	$\rightarrow \text{we}$	-2	$\frac{1}{4}$
	$\rightarrow \text{sushi}$	-3	$\frac{1}{8}$
	$\rightarrow \text{chopsticks}$	-3	$\frac{1}{8}$
PP	$\rightarrow \text{IN NP}$	0	1
IN	$\rightarrow \text{with}$	0	1
VP	$\rightarrow \text{V NP}$	-1	$\frac{1}{2}$
	$\rightarrow \text{VP PP}$	-2	$\frac{1}{4}$
	$\rightarrow \text{MD V}$	-2	$\frac{1}{4}$
V	$\rightarrow \text{eat}$	0	1

Table 10.2: An example weighted context-free grammar (WCFG). The weights are chosen so that $\exp \psi(\cdot)$ sums to one over right-hand sides for each non-terminal; this is required by probabilistic context-free grammars, but not by WCFGs in general.

Example Consider the weighted grammar shown in Table 10.2, and the analysis in Figure 10.2b.

$$\begin{aligned} \Psi(\tau) = & \psi(S \rightarrow \text{NP VP}) + \psi(VP \rightarrow \text{VP PP}) + \psi(VP \rightarrow \text{V NP}) + \psi(PP \rightarrow \text{IN NP}) \\ & + \psi(\text{NP} \rightarrow \text{We}) + \psi(\text{V} \rightarrow \text{eat}) + \psi(\text{NP} \rightarrow \text{sushi}) + \psi(\text{IN} \rightarrow \text{with}) + \psi(\text{NP} \rightarrow \text{chopsticks}) \end{aligned} \quad [10.6]$$

$$= 0 - 2 - 1 + 0 - 2 + 0 - 3 + 0 - 3 = -11. \quad [10.7]$$

5164 In the alternative parse in Figure 10.2a, the production $VP \rightarrow VP PP$ (with score -2) is
 5165 replaced with the production $NP \rightarrow NP PP$ (with score -1); all other productions are the
 5166 same. As a result, the score for this parse is -10.

5167 This example hints at a big problem with WCFG parsing on non-terminals such as
 5168 NP, VP, and PP: a WCFG will *always* prefer either VP or NP attachment, without regard
 5169 to what is being attached! This problem is addressed in § 10.5.

5170 10.3.1 Parsing with weighted context-free grammars

5171 The optimization problem in Equation 10.4 can be solved by modifying the CKY algo-
 5172 rithm. In the deterministic CKY algorithm, each cell $t[i, j]$ stored a set of non-terminals
 5173 capable of deriving the span $w_{i+1:j}$. We now augment the table so that the cell $t[i, j, X]$
 5174 is the *score of the best derivation of $w_{i+1:j}$ from non-terminal X* . This score is computed
 5175 recursively: for the anchored binary production $(X \rightarrow Y Z, (i, j, k))$, we compute:

Algorithm 14 CKY algorithm for parsing a string $w \in \Sigma^*$ in a weighted context-free grammar (N, Σ, R, S) , where N is the set of non-terminals and R is the set of weighted productions. The grammar is assumed to be in Chomsky normal form (§ 9.2.1.2). The function TRACEBACK is defined in Algorithm 13.

```

procedure WCKY( $w, G = (N, \Sigma, R, S)$ )
  for all  $i, j, X$  do ▷ Initialization
     $t[i, j, X] \leftarrow 0$ 
     $b[i, j, X] \leftarrow \emptyset$ 
  for  $m \in \{1, 2, \dots, M\}$  do
    for all  $X \in N$  do
       $t[m, m + 1, X] \leftarrow \psi(X \rightarrow w_m, (m, m + 1, m))$ 
  for  $\ell \in \{2, 3, \dots, M\}$  do
    for  $m \in \{0, 1, \dots, M - \ell\}$  do
      for  $k \in \{m + 1, m + 2, \dots, m + \ell - 1\}$  do
         $t[m, m + \ell, X] \leftarrow \max_{k, Y, Z} \psi(X \rightarrow Y Z, (m, m + \ell, k)) + t[m, k, Y] + t[k, m + \ell, Z]$ 
         $b[m, m + \ell, X] \leftarrow \operatorname{argmax}_{k, Y, Z} \psi(X \rightarrow Y Z, (m + \ell, k)) + t[m, k, Y] + t[k, m + \ell, Z]$ 
  return TRACEBACK( $S, 0, M, b$ )

```

- 5176 • the score of the anchored production, $\psi(X \rightarrow Y Z, (i, j, k))$;
 5177 • the score of the best derivation of the left child, $t[i, k, Y]$;
 5178 • the score of the best derivation of the right child, $t[k, j, Z]$.

5179 These scores are combined by addition. As in the unscored CKY algorithm, the table
 5180 is constructed by considering spans of increasing length, so the scores for spans $t[i, k, Y]$
 5181 and $t[k, j, Z]$ are guaranteed to be available at the time we compute the score $t[i, j, X]$. The
 5182 value $t[0, M, S]$ is the score of the best derivation of w from the grammar. Algorithm 14
 5183 formalizes this procedure.

5184 As in unweighted CKY, the parse is recovered from the table of back pointers b , where
 5185 each $b[i, j, X]$ stores the argmax split point k and production $X \rightarrow Y Z$ in the derivation of
 5186 $w_{i+1:j}$ from X . The best parse can be obtained by tracing these pointers backwards from
 5187 $b[0, M, S]$, all the way to the terminal symbols. This is analogous to the computation of the
 5188 best sequence of labels in the Viterbi algorithm by tracing pointers backwards from the
 5189 end of the trellis. Note that we need only store back-pointers for the *best* path to $t[i, j, X]$;
 5190 this follows from the locality assumption that the global score for a parse is a combination
 5191 of the local scores of each production in the parse.

Example Let's revisit the parsing table in Figure 10.1. In a weighted CFG, each cell would include a score for each non-terminal; non-terminals that cannot be generated are

Algorithm 15 Generative model for derivations from probabilistic context-free grammars in Chomsky Normal Form (CNF).

```

procedure DRAWSUBTREE(X)
    sample  $(X \rightarrow \alpha) \sim p(\alpha | X)$ 
    if  $\alpha = (Y Z)$  then
        return DRAWSUBTREE(Y)  $\cup$  DRAWSUBTREE(Z)
    else
        return  $(X \rightarrow \alpha)$             $\triangleright$  In CNF, all unary productions yield terminal symbols

```

assumed to have a score of $-\infty$. The first diagonal contains the scores of unary productions: $t[0, 1, \text{NP}] = -2$, $t[1, 2, \text{V}] = 0$, and so on. At the next diagonal, we compute the scores for spans of length 2: $t[1, 3, \text{VP}] = -1 + 0 - 3 = -4$, $t[3, 5, \text{PP}] = 0 + 0 - 3 = -3$, and so on. Things get interesting when we reach the cell $t[1, 5, \text{VP}]$, which contains the score for the derivation of the span $w_{2:5}$ from the non-terminal VP. This score is computed as a max over two alternatives,

$$t[1, 5, \text{VP}] = \max(\psi(\text{VP} \rightarrow \text{VP PP}, (1, 3, 5)) + t[1, 3, \text{VP}] + t[3, 5, \text{PP}], \\ \psi(\text{VP} \rightarrow \text{V NP}, (1, 2, 5)) + t[1, 2, \text{V}] + t[2, 5, \text{NP}]) \quad [10.8]$$

$$= \max(-2 - 4 - 3, -1 + 0 - 7) = -8. \quad [10.9]$$

5192 Since the second case is the argmax, we set the back-pointer $b[1, 5, \text{VP}] = (\text{V}, \text{NP}, 2)$, enabling the optimal derivation to be recovered.

5194 **10.3.2 Probabilistic context-free grammars**

5195 **Probabilistic context-free grammars (PCFGs)** are a special case of weighted context-
5196 free grammars that arises when the weights correspond to probabilities. Specifically, the
5197 weight $\psi(X \rightarrow \alpha, (i, j, k)) = \log p(\alpha | X)$, where the probability of the right-hand side
5198 α is conditioned on the non-terminal X . These probabilities must be normalized over all
5199 possible right-hand sides, so that $\sum_\alpha p(\alpha | X) = 1$, for all X . For a given parse τ , the prod-
5200 uct of the probabilities of the productions is equal to $p(\tau)$, under the **generative model**
5201 $\tau \sim \text{DRAWSUBTREE}(S)$, where the function DRAWSUBTREE is defined in Algorithm 15.

5202 The conditional probability of a parse given a string is,

$$p(\tau | w) = \frac{p(\tau)}{\sum_{\tau' \in \mathcal{T}(w)} p(\tau')} = \frac{\exp \Psi(\tau)}{\sum_{\tau' \in \mathcal{T}(w)} \exp \Psi(\tau')}, \quad [10.10]$$

5203 where $\Psi(\tau) = \sum_{X \rightarrow \alpha, (i, j, k) \in \tau} \psi(X \rightarrow \alpha)$; the anchor is ignored. Because the probability
5204 is monotonic in the score $\Psi(\tau)$, the maximum likelihood parse can be identified by the
5205 CKY algorithm without modification. If a normalized probability $p(\tau | w)$ is required,
5206 the denominator of Equation 10.10 can be computed by the **inside recurrence**, described
5207 below.

Example The WCFG in Table 10.2 is designed so that the weights are log-probabilities, satisfying the constraint $\sum_{\alpha} \exp \psi(X \rightarrow \alpha) = 1$. As noted earlier, there are two parses in $\mathcal{T}(\text{we eat sushi with chopsticks})$, with scores $\Psi(\tau_1) = \log p(\tau_1) = -10$ and $\Psi(\tau_2) = \log p(\tau_2) = -11$. Therefore, the conditional probability $p(\tau_1 | \mathbf{w})$ is equal to,

$$p(\tau_1 | \mathbf{w}) = \frac{p(\tau_1)}{p(\tau_1) + p(\tau_2)} = \frac{\exp \Psi(\tau_1)}{\exp \Psi(\tau_1) + \exp \Psi(\tau_2)} = \frac{2^{-10}}{2^{-10} + 2^{-11}} = \frac{2}{3}. \quad [10.11]$$

5208 **The inside recurrence** The denominator of Equation 10.10 can be viewed as a language
5209 model, summing over all valid derivations of the string \mathbf{w} ,

$$p(\mathbf{w}) = \sum_{\tau': \text{yield}(\tau') = \mathbf{w}} p(\tau'). \quad [10.12]$$

Just as the CKY algorithm makes it possible to maximize over all such analyses, with a few modifications it can also compute their sum. Each cell $t[i, j, X]$ must store the log probability of deriving $\mathbf{w}_{i+1:j}$ from non-terminal X . To compute this, we replace the maximization over split points k and productions $X \rightarrow Y Z$ with a “log-sum-exp” operation, which exponentiates the log probabilities of the production and the children, sums them in probability space, and then converts back to the log domain:

$$t[i, j, X] = \log \sum_{k, Y, Z} \exp (\psi(X \rightarrow Y Z) + t[i, k, Y] + t[k, j, Z]) \quad [10.13]$$

$$= \log \sum_{k, Y, Z} \exp (\log p(Y Z | X) + \log p(Y \rightarrow \mathbf{w}_{i+1:k}) + \log p(Z \rightarrow \mathbf{w}_{k+1:j})) \quad [10.14]$$

$$= \log \sum_{k, Y, Z} p(Y Z | X) \times p(Y \rightarrow \mathbf{w}_{i+1:k}) \times p(Z \rightarrow \mathbf{w}_{k+1:j}) \quad [10.15]$$

$$= \log \sum_{k, Y, Z} p(Y Z, \mathbf{w}_{i+1:k}, \mathbf{w}_{k+1:j} | X) \quad [10.16]$$

$$= \log p(X \rightarrow \mathbf{w}_{i+1:j}). \quad [10.17]$$

5210 This is called the **inside recurrence**, because it computes the probability of each subtree
5211 as a combination of the probabilities of the smaller subtrees that are inside of it. The
5212 name implies a corresponding **outside recurrence**, which computes the probability of
5213 a non-terminal X spanning $\mathbf{w}_{i+1:j}$, joint with the outside context $(\mathbf{w}_{1:i}, \mathbf{w}_{j+1:M})$. This
5214 recurrence is described in § 10.4.3. The inside and outside recurrences are analogous to the
5215 forward and backward recurrences in probabilistic sequence labeling (see § 7.5.3.3). They
5216 can be used to compute the marginal probabilities of individual anchored productions,
5217 $p(X \rightarrow \alpha, (i, j, k) | \mathbf{w})$, summing over all possible derivations of \mathbf{w} .

5218 **10.3.3 *Semiring weighted context-free grammars**

The weighted and unweighted CKY algorithms can be unified with the inside recurrence using the same semiring notation described in § 7.7.3. The generalized recurrence is:

$$t[i, j, X] = \bigoplus_{k, Y, Z} \psi(X \rightarrow Y Z, (i, j, k)) \otimes t[i, k, Y] \otimes t[k, j, Z]. \quad [10.18]$$

5219 This recurrence subsumes all of the algorithms that we have encountered in this chapter.

5220 **Unweighted CKY.** When $\psi(X \rightarrow \alpha, (i, j, k))$ is a *Boolean truth value* $\{\top, \perp\}$, \otimes is logical
5221 conjunction, and \bigoplus is logical disjunction, then we derive the CKY recurrence for
5222 unweighted context-free grammars, discussed in § 10.1 and Algorithm 13.

5223 **Weighted CKY.** When $\psi(X \rightarrow \alpha, (i, j, k))$ is a scalar score, \otimes is addition, and \bigoplus is maxi-
5224 maximization, then we derive the CKY recurrence for weighted context-free grammars,
5225 discussed in § 10.3 and Algorithm 14. When $\psi(X \rightarrow \alpha, (i, j, k)) = \log p(\alpha \mid X)$,
5226 this same setting derives the CKY recurrence for finding the maximum likelihood
5227 derivation in a probabilistic context-free grammar.

5228 **Inside recurrence.** When $\psi(X \rightarrow \alpha, (i, j, k))$ is a log probability, \otimes is addition, and $\bigoplus =$
5229 $\log \sum \exp$, then we derive the inside recurrence for probabilistic context-free gram-
5230 mmars, discussed in § 10.3.2. It is also possible to set $\psi(X \rightarrow \alpha, (i, j, k))$ directly equal
5231 to the probability $p(\alpha \mid X)$. In this case, \otimes is multiplication, and \bigoplus is addition.
5232 While this may seem more intuitive than working with log probabilities, there is the
5233 risk of underflow on long inputs.

5234 Regardless of how the scores are combined, the key point is the locality assumption:
5235 the score for a derivation is the combination of the independent scores for each anchored
5236 production, and these scores do not depend on any other part of the derivation. For exam-
5237 ple, if two non-terminals are siblings, the scores of productions from these non-terminals
5238 are computed independently. This locality assumption is analogous to the first-order
5239 Markov assumption in sequence labeling, where the score for transitions between tags
5240 depends only on the previous tag and current tag, and not on the history. As with se-
5241 quence labeling, this assumption makes it possible to find the optimal parse efficiently; its
5242 linguistic limitations are discussed in § 10.5.

5243 **10.4 Learning weighted context-free grammars**

5244 Like sequence labeling, context-free parsing is a form of structure prediction. As a result,
5245 WCFGs can be learned using the same set of algorithms: generative probabilistic models,
5246 structured perceptron, maximum conditional likelihood, and maximum margin learning.

5247 In all cases, learning requires a **treebank**, which is a dataset of sentences labeled with
 5248 context-free parses. Parsing research was catalyzed by the **Penn Treebank** (Marcus et al.,
 5249 1993), the first large-scale dataset of this type (see § 9.2.2). Phrase structure treebanks exist
 5250 for roughly two dozen other languages, with coverage mainly restricted to European and
 5251 East Asian languages, plus Arabic and Urdu.

5252 **10.4.1 Probabilistic context-free grammars**

Probabilistic context-free grammars are similar to hidden Markov models, in that they are generative models of text. In this case, the parameters of interest correspond to probabilities of productions, conditional on the left-hand side. As with hidden Markov models, these parameters can be estimated by relative frequency:

$$\psi(X \rightarrow \alpha) = \log p(X \rightarrow \alpha) \quad [10.19]$$

$$\hat{p}(X \rightarrow \alpha) = \frac{\text{count}(X \rightarrow \alpha)}{\text{count}(X)}. \quad [10.20]$$

5253 For example, the probability of the production $\text{NP} \rightarrow \text{DET NN}$ is the corpus count of
 5254 this production, divided by the count of the non-terminal NP. This estimator applies
 5255 to terminal productions as well: the probability of $\text{NN} \rightarrow \text{whale}$ is the count of how often
 5256 *whale* appears in the corpus as generated from an NN tag, divided by the total count of the
 5257 NN tag. Even with the largest treebanks — currently on the order of one million tokens
 5258 — it is difficult to accurately compute probabilities of even moderately rare events, such
 5259 as $\text{NN} \rightarrow \text{whale}$. Therefore, smoothing is critical for making PCFGs effective.

5260 **10.4.2 Feature-based parsing**

5261 The scores for each production can be computed as an inner product of weights and fea-
 5262 tures,

$$\psi(X \rightarrow \alpha) = \boldsymbol{\theta} \cdot \mathbf{f}(X, \alpha, (i, j, k), \mathbf{w}), \quad [10.21]$$

5263 where the feature vector $\mathbf{f}(X, \alpha)$ is a function of the left-hand side X , the right-hand side
 5264 α , the anchor indices (i, j, k) , and the input \mathbf{w} .

5265 The basic feature $\mathbf{f}(X, \alpha, (i, j, k)) = \{(X, \alpha)\}$ encodes only the identity of the pro-
 5266 duction itself, which is a discriminatively-trained model with the same expressiveness as
 5267 a PCFG. Features on anchored productions can include the words that border the span
 5268 w_i, w_{j+1} , the word at the split point w_{k+1} , the presence of a verb or noun in the left child
 5269 span $w_{i+1:k}$, and so on (Durrett and Klein, 2015). Scores on anchored productions can be
 5270 incorporated into CKY parsing without any modification to the algorithm, because it is
 5271 still possible to compute each element of the table $t[i, j, X]$ recursively from its immediate
 5272 children.

5273 Other features can be obtained by grouping elements on either the left-hand or right-
 5274 hand side: for example it can be particularly beneficial to compute additional features
 5275 by clustering terminal symbols, with features corresponding to groups of words with
 5276 similar syntactic properties. The clustering can be obtained from unlabeled datasets that
 5277 are much larger than any treebank, improving coverage. Such methods are described in
 5278 chapter 14.

Feature-based parsing models can be estimated using the usual array of discriminative learning techniques. For example, a structure perceptron update can be computed as (Carreras et al., 2008),

$$\mathbf{f}(\tau, \mathbf{w}^{(i)}) = \sum_{(X \rightarrow \alpha, (i, j, k)) \in \tau} \mathbf{f}(X, \alpha, (i, j, k), \mathbf{w}^{(i)}) \quad [10.22]$$

$$\hat{\tau} = \operatorname{argmax}_{\tau \in \mathcal{T}(\mathbf{w})} \mathbf{f}(\tau, \mathbf{w}^{(i)}) \quad [10.23]$$

$$\boldsymbol{\theta} \leftarrow \mathbf{f}(\tau^{(i)}, \mathbf{w}^{(i)}) - \mathbf{f}(\hat{\tau}, \mathbf{w}^{(i)}). \quad [10.24]$$

5279 A margin-based objective can be optimized by selecting $\hat{\tau}$ through cost-augmented decoding (§ 2.3.2), enforcing a margin of $\Delta(\hat{\tau}, \tau)$ between the hypothesis and the reference parse,
 5280 where Δ is a non-negative cost function, such as the Hamming loss (Stern et al., 2017). It
 5281 is also possible to train feature-based parsing models by conditional log-likelihood, as
 5282 described in the next section.

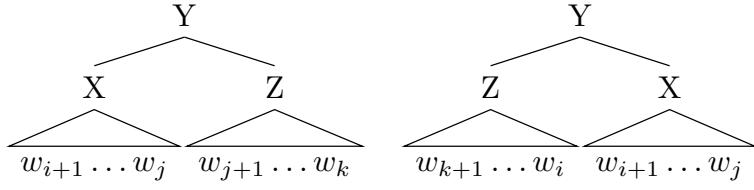
5284 10.4.3 *Conditional random field parsing

5285 The score of a derivation $\Psi(\tau)$ can be converted into a probability by normalizing over all
 5286 possible derivations,

$$p(\tau | \mathbf{w}) = \frac{\exp \Psi(\tau)}{\sum_{\tau' \in \mathcal{T}(\mathbf{w})} \exp \Psi(\tau')}. \quad [10.25]$$

5287 Using this probability, a WCFG can be trained by maximizing the conditional log-likelihood
 5288 of a labeled corpus.

5289 Just as in logistic regression and the conditional random field over sequences, the
 5290 gradient of the conditional log-likelihood is the difference between the observed and ex-
 5291 pected counts of each feature. The expectation $E_{\tau|\mathbf{w}}[\mathbf{f}(\tau, \mathbf{w}^{(i)}); \boldsymbol{\theta}]$ requires summing over
 5292 all possible parses, and computing the marginal probabilities of anchored productions,
 5293 $p(X \rightarrow \alpha, (i, j, k) | \mathbf{w})$. In CRF sequence labeling, marginal probabilities over tag bigrams
 5294 are computed by the two-pass **forward-backward algorithm** (§ 7.5.3.3). The analogue for
 5295 context-free grammars is the **inside-outside algorithm**, in which marginal probabilities
 5296 are computed from terms generated by an upward and downward pass over the parsing
 5297 chart:

Figure 10.3: The two cases faced by the outside recurrence in the computation of $\beta(i, j, X)$

- The upward pass is performed by the **inside recurrence**, which is described in § 10.3.2. Each inside variable $\alpha(i, j, X)$ is the score of deriving $w_{i+1:j}$ from the non-terminal X . In a PCFG, this corresponds to the log-probability $\log p(w_{i+1:j} \mid X)$. This is computed by the recurrence,

$$\alpha(i, j, X) \triangleq \log \sum_{(X \rightarrow Y \ Z)} \sum_{k=i+1}^j \exp (\psi(X \rightarrow Y \ Z, (i, j, k)) + \alpha(i, k, Y) + \alpha(k, j, Z)). \quad [10.26]$$

5298 The initial condition of this recurrence is $\alpha(m - 1, m, X) = \psi(X \rightarrow w_m)$. The de-
5299 nominator $\sum_{\tau \in \mathcal{T}(w)} \exp \Psi(\tau)$ is equal to $\exp \alpha(0, M, S)$.

- The downward pass is performed by the **outside recurrence**, which recursively populates the same table structure, starting at the root of the tree. Each outside variable $\beta(i, j, X)$ is the score of having a phrase of type X covering the span $(i + 1 : j)$, joint with the exterior context $w_{1:i}$ and $w_{j+1:M}$. In a PCFG, this corresponds to the log probability $\log p((X, i + 1, j), w_{1:i}, w_{j+1:M})$. Each outside variable is computed by the recurrence,

$$\exp \beta(i, j, X) \triangleq \sum_{(Y \rightarrow X \ Z)} \sum_{k=j+1}^M \exp [\psi(Y \rightarrow X \ Z, (i, k, j)) + \alpha(j, k, Z) + \beta(i, k, Y)] \quad [10.27]$$

$$+ \sum_{(Y \rightarrow Z \ X)} \sum_{k=0}^{i-1} \exp [\psi(Y \rightarrow Z \ X, (k, i, j)) + \alpha(k, i, Z) + \beta(k, j, Y)]. \quad [10.28]$$

5300 The first line of Equation 10.28 is the score under the condition that X is a left child
5301 of its parent, which spans $w_{i+1:k}$, with $k > j$; the second line is the score under the
5302 condition that X is a right child of its parent Y , which spans $w_{k+1:j}$, with $k < i$.
5303 The two cases are shown in Figure 10.3. In each case, we sum over all possible
5304 productions with X on the right-hand side. The parent Y is bounded on one side

5305 by either i or j , depending on whether X is a left or right child of Y ; we must sum
 5306 over all possible values for the other boundary. The initial conditions for the outside
 5307 recurrence are $\beta(0, M, S) = 0$ and $\beta(0, M, X \neq S) = -\infty$.

The marginal probability of a non-terminal X over span $w_{i+1:j}$ is written $p(X \rightsquigarrow w_{i+1:j} | w)$, and can be computed from the inside and outside scores,

$$p(X \rightsquigarrow w_{i+1:j} | w) = \frac{p(X \rightsquigarrow w_{i+1:j}, w)}{p(w)} \quad [10.29]$$

$$= \frac{p(w_{i+1:j} | X) \times p(X, w_{1:i}, w_{j+1:M})}{p(w)} \quad [10.30]$$

$$= \frac{\exp(\alpha(i, j, X) + \beta(i, j, X))}{\exp \alpha(0, M, S)}. \quad [10.31]$$

5308 Marginal probabilities of individual productions can be computed similarly (see exercise
 5309 2). These marginal probabilities can be used for training a conditional random field parser,
 5310 and also for the task of unsupervised **grammar induction**, in which a PCFG is estimated
 5311 from a dataset of unlabeled text (Lari and Young, 1990; Pereira and Schabes, 1992).

5312 10.4.4 Neural context-free grammars

5313 Recent work has applied neural representations to parsing, representing each span with
 5314 a dense numerical vector (Socher et al., 2013; Durrett and Klein, 2015; Cross and Huang,
 5315 2016).⁴ For example, the anchor (i, j, k) and sentence w can be associated with a fixed-
 5316 length column vector,

$$\mathbf{v}_{(i,j,k)} = [\mathbf{u}_{w_{i-1}}; \mathbf{u}_{w_i}; \mathbf{u}_{w_{j-1}}; \mathbf{u}_{w_j}; \mathbf{u}_{w_{k-1}}; \mathbf{u}_{w_k}], \quad [10.32]$$

where \mathbf{u}_{w_i} is a word embedding associated with the word w_i . The vector $\mathbf{v}_{i,j,k}$ can then be passed through a feedforward neural network, and used to compute the score of the anchored production. For example, this score can be computed as a bilinear product (Durrett and Klein, 2015),

$$\tilde{\mathbf{v}}_{(i,j,k)} = \text{FeedForward}(\mathbf{v}_{(i,j,k)}) \quad [10.33]$$

$$\psi(X \rightarrow \alpha, (i, j, k)) = \tilde{\mathbf{v}}_{(i,j,k)}^\top \Theta \mathbf{f}(X \rightarrow \alpha), \quad [10.34]$$

5317 where $\mathbf{f}(X \rightarrow \alpha)$ is a vector of discrete features of the production, and Θ is a parameter
 5318 matrix. The matrix Θ and the parameters of the feedforward network can be learned by
 5319 backpropagating from an objective such as the margin loss or the negative conditional
 5320 log-likelihood.

⁴Earlier work on neural constituent parsing used transition-based parsing algorithms (§ 10.6.2) rather than CKY-style chart parsing (Henderson, 2004; Titov and Henderson, 2007).

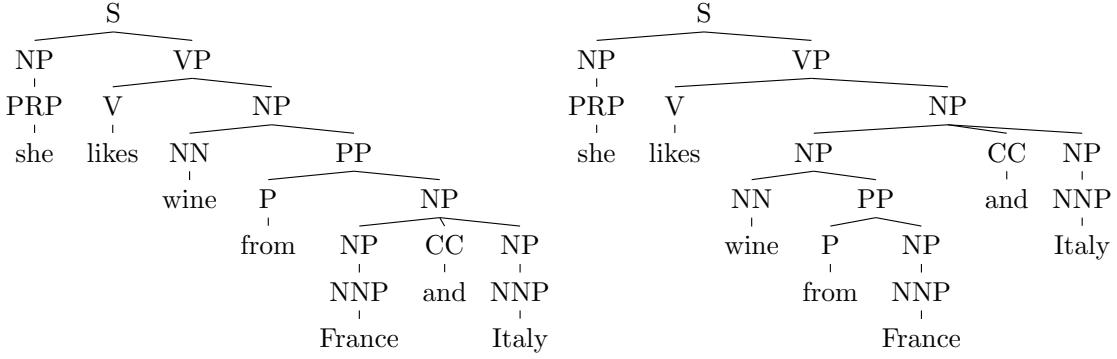


Figure 10.4: The left parse is preferable because of the conjunction of phrases headed by *France* and *Italy*, but these parses cannot be distinguished by a WCFG.

5321 10.5 Grammar refinement

5322 The locality assumptions underlying CFG parsing depend on the granularity of the non-
 5323 terminals. For the Penn Treebank non-terminals, there are several reasons to believe that
 5324 these assumptions are too strong to enable accurate parsing (Johnson, 1998):

- 5325 • The context-free assumption is too strict: for example, the probability of the produc-
 5326 tion $NP \rightarrow NP\ PP$ is much higher (in the PTB) if the parent of the noun phrase is a
 5327 verb phrase (indicating that the NP is a direct object) than if the parent is a sentence
 5328 (indicating that the NP is the subject of the sentence).
- 5329 • The Penn Treebank non-terminals are too coarse: there are many kinds of noun
 5330 phrases and verb phrases, and accurate parsing sometimes requires knowing the
 5331 difference. As we have already seen, when faced with prepositional phrase at-
 5332 tachment ambiguity, a weighted CFG will either always choose NP attachment (if
 5333 $\psi(NP \rightarrow NP\ PP) > \psi(VP \rightarrow VP\ PP)$), or it will always choose VP attachment. To
 5334 get more nuanced behavior, more fine-grained non-terminals are needed.
- 5335 • More generally, accurate parsing requires some amount of **semantics** — understand-
 5336 ing the meaning of the text to be parsed. Consider the example *cats scratch people with*
 5337 *claws*: knowledge of about *cats*, *claws*, and scratching is necessary to correctly resolve
 5338 the attachment ambiguity.

5339 An extreme example is shown in Figure 10.4. The analysis on the left is preferred
 5340 because of the conjunction of similar entities *France* and *Italy*. But given the non-terminals
 5341 shown in the analyses, there is no way to differentiate these two parses, since they include
 5342 exactly the same productions. What is needed seems to be more precise non-terminals.
 5343 One possibility would be to rethink the linguistics behind the Penn Treebank, and ask

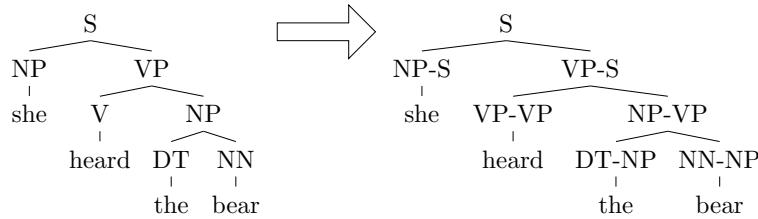


Figure 10.5: Parent annotation in a CFG derivation

5344 the annotators to try again. But the original annotation effort took five years, and there
 5345 is a little appetite for another annotation effort of this scope. Researchers have therefore
 5346 turned to automated techniques.

5347 10.5.1 Parent annotations and other tree transformations

The key assumption underlying context-free parsing is that productions depend only on the identity of the non-terminal on the left-hand side, and not on its ancestors or neighbors. The validity of this assumption is an empirical question, and it depends on the non-terminals themselves: ideally, every noun phrase (and verb phrase, etc) would be distributionally identical, so the assumption would hold. But in the Penn Treebank, the observed probability of productions often depends on the parent of the left-hand side. For example, noun phrases are more likely to be modified by prepositional phrases when they are in the object position (e.g., *they amused the students from Georgia*) than in the subject position (e.g., *the students from Georgia amused them*). This means that the $\text{NP} \rightarrow \text{NP PP}$ production is more likely if the entire constituent is the child of a VP than if it is the child of S. The observed statistics are (Johnson, 1998):

$$\Pr(\text{NP} \rightarrow \text{NP PP}) = 11\% \quad [10.35]$$

$$\Pr(\text{NP under S} \rightarrow \text{NP PP}) = 9\% \quad [10.36]$$

$$\Pr(\text{NP under VP} \rightarrow \text{NP PP}) = 23\%. \quad [10.37]$$

5348 This phenomenon can be captured by **parent annotation** (Johnson, 1998), in which each
 5349 non-terminal is augmented with the identity of its parent, as shown in Figure 10.5). This is
 5350 sometimes called **vertical Markovization**, since a Markov dependency is introduced be-
 5351 tween each node and its parent (Klein and Manning, 2003). It is analogous to moving from
 5352 a bigram to a trigram context in a hidden Markov model. In principle, parent annotation
 5353 squares the size of the set of non-terminals, which could make parsing considerably less
 5354 efficient. But in practice, the increase in the number of non-terminals that actually appear
 5355 in the data is relatively modest (Johnson, 1998).

5356 Parent annotation weakens the WCFG locality assumptions. This improves accuracy
 5357 by enabling the parser to make more fine-grained distinctions, which better capture real
 5358 linguistic phenomena. However, each production is more rare, and so careful smoothing
 5359 or regularization is required to control the variance over production scores.

5360 10.5.2 Lexicalized context-free grammars

5361 The examples in § 10.2.2 demonstrate the importance of individual words in resolving
 5362 parsing ambiguity: the preposition *on* is more likely to attach to *met*, while the preposition
 5363 *of* is more likely to attachment to *President*. But of all word pairs, which are relevant to
 5364 attachment decisions? Consider the following variants on the original examples:

- 5365 (10.4) We met the President of Mexico.
- 5366 (10.5) We met the first female President of Mexico.
- 5367 (10.6) They had supposedly met the President on Monday.

5368 The underlined words are the **head words** of their respective phrases: *met* heads the verb
 5369 phrase, and *President* heads the direct object noun phrase. These heads provide useful
 5370 semantic information. But they break the context-free assumption, which states that the
 5371 score for a production depends only on the parent and its immediate children, and not
 5372 the substructure under each child.

The incorporation of head words into context-free parsing is known as **lexicalization**,
 and is implemented in rules of the form,

$$\text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(of) \quad [10.38]$$

$$\text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(on). \quad [10.39]$$

5373 Lexicalization was a major step towards accurate PCFG parsing. It requires solving three
 5374 problems: identifying the heads of all constituents in a treebank; parsing efficiently while
 5375 keeping track of the heads; and estimating the scores for lexicalized productions.

5376 10.5.2.1 Identifying head words

5377 The head of a constituent is the word that is the most useful for determining how that
 5378 constituent is integrated into the rest of the sentence.⁵ The head word of a constituent is
 5379 determined recursively: for any non-terminal production, the head of the left-hand side
 5380 must be the head of one of the children. The head is typically selected according to a set of
 5381 deterministic rules, sometimes called **head percolation rules**. In many cases, these rules
 5382 are straightforward: the head of a noun phrase in a $\text{NP} \rightarrow \text{DET NN}$ production is the head

⁵This is a pragmatic definition, befitting our goal of using head words to improve parsing; for a more formal definition, see (Bender, 2013, chapter 7).

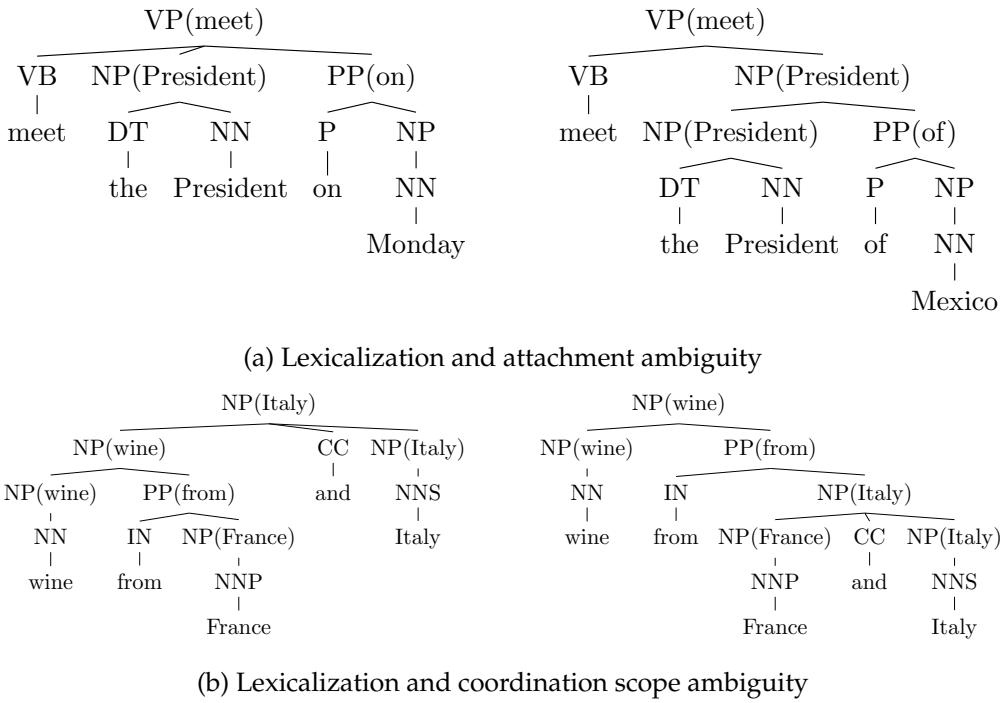


Figure 10.6: Examples of lexicalization

5383 of the noun; the head of a sentence in a $S \rightarrow NP\ VP$ production is the head of the verb
 5384 phrase.

5385 Table 10.3 shows a fragment of the head percolation rules used in many English pars-
 5386 ing systems. The meaning of the first rule is that to find the head of an S constituent, first
 5387 look for the rightmost VP child; if you don't find one, then look for the rightmost $SBAR$
 5388 child, and so on down the list. Verb phrases are headed by left verbs (the head of *can plan*
 5389 *on walking* is *planned*, since the modal verb *can* is tagged *MD*); noun phrases are headed by
 5390 the rightmost noun-like non-terminal (so the head of *the red cat* is *cat*),⁶ and prepositional
 5391 phrases are headed by the preposition (the head of *at Georgia Tech* is *at*). Some of these
 5392 rules are somewhat arbitrary — there's no particular reason why the head of *cats and dogs*
 5393 should be *dogs* — but the point here is just to get some lexical information that can support
 5394 parsing, not to make deep claims about syntax. Figure 10.6 shows the application of these
 5395 rules to two of the running examples.

⁶The noun phrase non-terminal is sometimes treated as a special case. Collins (1997) uses a heuristic that looks for the rightmost child which is a noun-like part-of-speech (e.g., *NN*, *NNP*), a possessive marker, or a superlative adjective (e.g., *the greatest*). If no such child is found, the heuristic then looks for the *leftmost* *NP*. If there is no child with tag *NP*, the heuristic then applies another priority list, this time from right to left.

Non-terminal	Direction	Priority
S	right	VP SBAR ADJP UCP NP
VP	left	VBD VBN MD VBZ TO VB VP VBG VBP ADJP NP
NP	right	N* EX \$ CD QP PRP ...
PP	left	IN TO FW

Table 10.3: A fragment of head percolation rules for English, from <http://www.cs.columbia.edu/~mcollins/papers/heads>

5396 10.5.2.2 Parsing lexicalized context-free grammars

5397 A naïve application of lexicalization would simply increase the set of non-terminals by
 5398 taking the cross-product with the set of terminal symbols, so that the non-terminals now
 5399 include symbols like $\text{NP}(\text{President})$ and $\text{VP}(\text{meet})$. Under this approach, the CKY parsing
 5400 algorithm could be applied directly to the lexicalized production rules. However, the
 5401 complexity would be cubic in the size of the vocabulary of terminal symbols, which would
 5402 clearly be intractable.

Another approach is to augment the CKY table with an additional index, keeping track of the head of each constituent. The cell $t[i, j, h, X]$ stores the score of the best derivation in which non-terminal X spans $w_{i+1:j}$ with head word h , where $i < h \leq j$. To compute such a table recursively, we must consider the possibility that each phrase gets its head from either its left or right child. The scores of the best derivations in which the head comes from the left and right child are denoted t_ℓ and t_r respectively, leading to the following recurrence:

$$t_\ell[i, j, h, X] = \max_{(X \rightarrow Y Z)} \max_{k > h} \max_{k < h' \leq j} t[i, k, h, Y] + t[k, j, h', Z] + \psi(X(h) \rightarrow Y(h)Z(h')) \quad [10.40]$$

$$t_r[i, j, h, X] = \max_{(X \rightarrow Y Z)} \max_{k < h} \max_{i < h' \leq k} t[i, k, h', Y] + t[k, j, h, Z] + (\psi(X(h) \rightarrow Y(h')Z(h))) \quad [10.41]$$

$$t[i, j, h, X] = \max(t_\ell[i, j, h, X], t_r[i, j, h, X]). \quad [10.42]$$

5403 To compute t_ℓ , we maximize over all split points $k > h$, since the head word must be in
 5404 the left child. We then maximize again over possible head words h' for the right child. An
 5405 analogous computation is performed for t_r . The size of the table is now $\mathcal{O}(M^3N)$, where
 5406 M is the length of the input and N is the number of non-terminals. Furthermore, each
 5407 cell is computed by performing $\mathcal{O}(M^2)$ operations, since we maximize over both the split
 5408 point k and the head h' . The time complexity of the algorithm is therefore $\mathcal{O}(RM^5N)$,
 5409 where R is the number of rules in the grammar. Fortunately, more efficient solutions are
 5410 possible. In general, the complexity of parsing can be reduced to $\mathcal{O}(M^4)$ in the length of

5411 the input; for a broad class of lexicalized CFGs, the complexity can be made cubic in the
 5412 length of the input, just as in unlexicalized CFGs (Eisner, 2000).

5413 **10.5.2.3 Estimating lexicalized context-free grammars**

5414 The final problem for lexicalized parsing is how to estimate weights for lexicalized pro-
 5415 ductions $X(i) \rightarrow Y(j) Z(k)$. These productions are said to be **bilexical**, because they
 5416 involve scores over pairs of words: in the example *meet the President of Mexico*, we hope
 5417 to choose the correct attachment point by modeling the bilexical affinities of (*meet, of*) and
 5418 (*President, of*). The number of such word pairs is quadratic in the size of the vocabulary,
 5419 making it difficult to estimate the weights of lexicalized production rules directly from
 5420 data. This is especially true for probabilistic context-free grammars, in which the weights
 5421 are obtained from smoothed relative frequency. In a treebank with a million tokens, a
 5422 vanishingly small fraction of the possible lexicalized productions will be observed more
 5423 than once.⁷ The Charniak (1997) and Collins (1997) parsers therefore focus on approxi-
 5424 mating the probabilities of lexicalized productions, using various smoothing techniques
 5425 and independence assumptions.

In discriminatively-trained weighted context-free grammars, the scores for each production can be computed from a set of features, which can be made progressively more fine-grained (Finkel et al., 2008). For example, the score of the lexicalized production $\text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(of)$ can be computed from the following features:

$$\begin{aligned} f(\text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(of)) = & \{\text{NP}(\text{*}) \rightarrow \text{NP}(\text{*}) \text{ PP}(\text{*}), \\ & \text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(\text{*}), \\ & \text{NP}(\text{*}) \rightarrow \text{NP}(\text{*}) \text{ PP}(of), \\ & \text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(of)\} \end{aligned}$$

5426 The first feature scores the unlexicalized production $\text{NP} \rightarrow \text{NP PP}$; the next two features
 5427 lexicalize only one element of the production, thereby scoring the appropriateness of NP
 5428 attachment for the individual words *President* and *of*; the final feature scores the specific
 5429 bilexical affinity of *President* and *of*. For bilexical pairs that are encountered frequently in
 5430 the treebank, this bilexical feature can play an important role in parsing; for pairs that are
 5431 absent or rare, regularization will drive its weight to zero, forcing the parser to rely on the
 5432 more coarse-grained features.

5433 In chapter 14, we will encounter techniques for clustering words based on their **distribu-**
 5434 **tional** properties — the contexts in which they appear. Such a clustering would group
 5435 rare and common words, such as *whale*, *shark*, *Leviathan*. Word clusters can be used

⁷The real situation is even more difficult, because non-binary context-free grammars can involve **trilexical** or higher-order dependencies, between the head of the constituent and multiple of its children (Carreras et al., 2008).

5436 as features in discriminative lexicalized parsing, striking a middle ground between full
 5437 lexicalization and non-terminals (Finkel et al., 2008). In this way, labeled examples con-
 5438 taining relatively common words like *whale* can help to improve parsing for rare words
 5439 like *beluga*, as long as those two words are clustered together.

5440 **10.5.3 *Refinement grammars**

5441 Lexicalization improves on context-free parsing by adding detailed information in the
 5442 form of lexical heads. However, estimating the scores of lexicalized productions is dif-
 5443 ficult. Klein and Manning (2003) argue that the right level of linguistic detail is some-
 5444 where between treebank categories and individual words. Some parts-of-speech and non-
 5445 terminals are truly substitutable: for example, *cat*/N and *dog*/N. But others are not: for
 5446 example, the preposition *of* exclusively attaches to nouns, while the preposition *as* is more
 5447 likely to modify verb phrases. Klein and Manning (2003) obtained a 2% improvement in
 5448 *F*-MEASURE on a parent-annotated PCFG parser by making a single change: splitting the
 5449 preposition category into six subtypes. They propose a series of linguistically-motivated
 5450 refinements to the Penn Treebank annotations, which in total yielded a 40% error reduc-
 5451 tion.

5452 Non-terminal refinement process can be automated by treating the refined categories
 5453 as latent variables. For example, we might split the noun phrase non-terminal into NP1, NP2, NP3, ...,
 5454 without defining in advance what each refined non-terminal corresponds to. This can
 5455 be treated as **partially supervised learning**, similar to the multi-component document
 5456 classification model described in § 5.2.3. A latent variable PCFG can be estimated by
 5457 expectation-maximization (Matsuzaki et al., 2005):

- 5458 • In the E-step, estimate a marginal distribution q over the refinement type of each
 5459 non-terminal in each derivation. These marginals are constrained by the original
 5460 annotation: an NP can be reannotated as NP4, but not as VP3. Marginal probabili-
 5461 ties over refined productions can be computed from the **inside-outside algorithm**,
 5462 as described in § 10.4.3, where the E-step enforces the constraints imposed by the
 5463 original annotations.
- 5464 • In the M-step, recompute the parameters of the grammar, by summing over the
 5465 probabilities of anchored productions that were computed in the E-step:

$$E[\text{count}(X \rightarrow Y Z)] = \sum_{i=0}^M \sum_{j=i}^M \sum_{k=i}^j p(X \rightarrow Y Z, (i, j, k) | \mathbf{w}). \quad [10.43]$$

5466 As usual, this process can be iterated to convergence. To determine the number of re-
 5467 finement types for each tag, Petrov et al. (2006) apply a split-merge heuristic; Liang et al.
 5468 (2007) and Finkel et al. (2007) apply **Bayesian nonparametrics** (Cohen, 2016).

Proper nouns			
NNP-14	<i>Oct.</i>	<i>Nov.</i>	<i>Sept.</i>
NNP-12	<i>John</i>	<i>Robert</i>	<i>James</i>
NNP-2	<i>J.</i>	<i>E.</i>	<i>L.</i>
NNP-1	<i>Bush</i>	<i>Noriega</i>	<i>Peters</i>
NNP-15	<i>New</i>	<i>San</i>	<i>Wall</i>
NNP-3	<i>York</i>	<i>Francisco</i>	<i>Street</i>
Personal Pronouns			
PRP-0	<i>It</i>	<i>He</i>	<i>I</i>
PRP-1	<i>it</i>	<i>he</i>	<i>they</i>
PRP-2	<i>it</i>	<i>them</i>	<i>him</i>

Table 10.4: Examples of automatically refined non-terminals and some of the words that they generate (Petrov et al., 2006).

5469 Some examples of refined non-terminals are shown in Table 10.4. The proper nouns
 5470 differentiate months, first names, middle initials, last names, first names of places, and
 5471 second names of places; each of these will tend to appear in different parts of grammatical
 5472 productions. The personal pronouns differentiate grammatical role, with PRP-0 appear-
 5473 ing in subject position at the beginning of the sentence (note the capitalization), PRP-1
 5474 appearing in subject position but not at the beginning of the sentence, and PRP-2 appear-
 5475 ing in object position.

5476 10.6 Beyond context-free parsing

5477 In the context-free setting, the score for a parse is a combination of the scores of individual
 5478 productions. As we have seen, these models can be improved by using finer-grained non-
 5479 terminals, via parent-annotation, lexicalization, and automated refinement. However, the
 5480 inherent limitations to the expressiveness of context-free parsing motivate the consider-
 5481 ation of other search strategies. These strategies abandon the optimality guaranteed by
 5482 bottom-up parsing, in exchange for the freedom to consider arbitrary properties of the
 5483 proposed parses.

5484 10.6.1 Reranking

5485 A simple way to relax the restrictions of context-free parsing is to perform a two-stage pro-
 5486 cess, in which a context-free parser generates a k -best list of candidates, and a **reranker**
 5487 then selects the best parse from this list (Charniak and Johnson, 2005; Collins and Koo,
 5488 2005). The reranker can be trained from an objective that is similar to multi-class classi-
 5489 fication: the goal is to learn weights that assign a high score to the reference parse, or to

5490 the parse on the k -best list that has the lowest error. In either case, the reranker need only
 5491 evaluate the K best parses, and so no context-free assumptions are necessary. This opens
 5492 the door to more expressive scoring functions:

- 5493 • It is possible to incorporate arbitrary non-local features, such as the structural par-
 5494 allelism and right-branching orientation of the parse (Charniak and Johnson, 2005).
 5495 • Reranking enables the use of **recursive neural networks**, in which each constituent
 5496 span $w_{i+1:j}$ receives a vector $\mathbf{u}_{i,j}$ which is computed from the vector representa-
 5497 tions of its children, using a composition function that is linked to the production
 5498 rule (Socher et al., 2013), e.g.,

$$\mathbf{u}_{i,j} = f \left(\Theta_{X \rightarrow Y} Z \begin{bmatrix} \mathbf{u}_{i,k} \\ \mathbf{u}_{k,j} \end{bmatrix} \right) \quad [10.44]$$

5499 The overall score of the parse can then be computed from the final vector, $\Psi(\tau) =$
 5500 $\theta \mathbf{u}_{0,M}$.

5501 Reranking can yield substantial improvements in accuracy. The main limitation is that it
 5502 can only find the best parse among the K -best offered by the generator, so it is inherently
 5503 limited by the ability of the bottom-up parser to find high-quality candidates.

5504 10.6.2 Transition-based parsing

5505 Structure prediction can be viewed as a form of search. An alternative to bottom-up pars-
 5506 ing is to read the input from left-to-right, gradually building up a parse structure through
 5507 a series of **transitions**. Transition-based parsing is described in more detail in the next
 5508 chapter, in the context of dependency parsing. However, it can also be applied to CFG
 5509 parsing, as briefly described here.

5510 For any context-free grammar, there is an equivalent **pushdown automaton**, a model
 5511 of computation that accepts exactly those strings that can be derived from the grammar.
 5512 This computational model consumes the input from left to right, while pushing and pop-
 5513 ping elements on a stack. This architecture provides a natural transition-based parsing
 5514 framework for context-free grammars, known as **shift-reduce parsing**.

5515 Shift-reduce parsing is a type of transition-based parsing, in which the parser can take
 5516 the following actions:

- 5517 • *shift* the next terminal symbol onto the stack;
 5518 • *unary-reduce* the top item on the stack, using a unary production rule in the gram-
 5519 mar;
 5520 • *binary-reduce* the top two items onto the stack, using a binary production rule in the
 5521 grammar.

5522 The set of available actions is constrained by the situation: the parser can only shift if
 5523 there are remaining terminal symbols in the input, and it can only reduce if an applicable
 5524 production rule exists in the grammar. If the parser arrives at a state where the input
 5525 has been completely consumed, and the stack contains only the element S, then the input
 5526 is accepted. If the parser arrives at a non-accepting state where there are no possible
 5527 actions, the input is rejected. A parse error occurs if there is some action sequence that
 5528 would accept an input, but the parser does not find it.

5529 **Example** Consider the input *we eat sushi* and the grammar in Table 10.1. The input can
 5530 be parsed through the following sequence of actions:

- 5531 1. **Shift** the first token *we* onto the stack.
- 5532 2. **Reduce** the top item on the stack to NP, using the production $NP \rightarrow we$.
- 5533 3. **Shift** the next token *eat* onto the stack, and **reduce** it to V with the production $V \rightarrow eat$.
- 5535 4. **Shift** the final token *sushi* onto the stack, and **reduce** it to NP. The input has been
 5536 completely consumed, and the stack contains [NP, V, NP].
- 5537 5. **Reduce** the top two items using the production $VP \rightarrow V NP$. The stack now con-
 5538 tains [VP, NP].
- 5539 6. **Reduce** the top two items using the production $S \rightarrow NP VP$. The stack now contains
 5540 [S]. Since the input is empty, this is an accepting state.

5541 One thing to notice from this example is that the number of shift actions is equal to the length of the input. The number of reduce actions is equal to the number of non-terminals in the analysis, which grows linearly in the length of the input. Thus, the overall time complexity of shift-reduce parsing is linear in the length of the input (assuming the complexity of each individual classification decision is constant in the length of the input).
 5546 This is far better than the cubic time complexity required by CKY parsing.

5547 **Transition-based parsing as inference** In general, it is not possible to guarantee that a transition-based parser will find the optimal parse, $\text{argmax}_\tau \Psi(\tau; \mathbf{w})$, even under the usual CFG independence assumptions. We could assign a score to each anchored parsing action in each context, with $\psi(a, c)$ indicating the score of performing action a in context c .
 5551 One might imagine that transition-based parsing could efficiently find the derivation that maximizes the sum of such scores. But this too would require backtracking and searching over an exponentially large number of possible action sequences: if a bad decision is made at the beginning of the derivation, then it may be impossible to recover the optimal action sequence without backtracking to that early mistake. This is known as a **search error**. Transition-based parsers can incorporate arbitrary features, without the restrictive

5557 independence assumptions required by chart parsing; search errors are the price that must
 5558 be paid for this flexibility.

5559 **Learning transition-based parsing** Transition-based parsing can be combined with ma-
 5560 chine learning by training a classifier to select the correct action in each situation. This
 5561 classifier is free to choose any feature of the input, the state of the parser, and the parse
 5562 history. However, there is no optimality guarantee: the parser may choose a suboptimal
 5563 parse, due to a mistake at the beginning of the analysis. Nonetheless, some of the strongest
 5564 CFG parsers are based on the shift-reduce architecture, rather than CKY. A recent genera-
 5565 tion of models links shift-reduce parsing with recurrent neural networks, updating a
 5566 hidden state vector while consuming the input (e.g., Cross and Huang, 2016; Dyer et al.,
 5567 2016). Learning algorithms for transition-based parsing are discussed in more detail in
 5568 § 11.3.

5569 **Exercises**

1. Consider the following PCFG:

$$p(X \rightarrow X X) = \frac{1}{2} \quad [10.45]$$

$$p(X \rightarrow Y) = \frac{1}{2} \quad [10.46]$$

$$p(Y \rightarrow \sigma) = \frac{1}{|\Sigma|}, \forall \sigma \in \Sigma \quad [10.47]$$

5570 a) Compute the probability $p(\hat{\tau})$ of the maximum probability parse for a string
 5571 $w \in \Sigma^M$.

5572 b) Compute the marginal probability $p(w) = \sum_{\tau: \text{yield}(\tau)=w} p(\tau)$.

5573 c) Compute the conditional probability $p(\hat{\tau} | w)$.

5574 2. Use the inside and outside scores to compute the marginal probability $p(X_{i:j} \rightarrow Y_{i:k-1} Z_{k:j} | w)$,
 5575 indicating that Y spans $w_{i:k-1}$, Z spans $w_{k:j}$, and X is the parent of Y and Z , span-
 5576 ning $w_{i:j}$.

5577 3. Suppose that the potentials $\Psi(X \rightarrow \alpha)$ are log-probabilities, so that $\sum_{\alpha} \exp \Psi(X \rightarrow \alpha) = 1$
 5578 for all X . Verify that the semiring inside recurrence from Equation 10.26 generates
 5579 the log-probability $\log p(w) = \log \sum_{\tau: \text{yield}(\tau)=w} p(\tau)$.

5580 4. more exercises tk

5581 Chapter 11

5582 Dependency parsing

5583 The previous chapter discussed algorithms for analyzing sentences in terms of nested con-
5584 stituents, such as noun phrases and verb phrases. However, many of the key sources of
5585 ambiguity in phrase-structure analysis relate to questions of **attachment**: where to attach a
5586 prepositional phrase or complement clause, how to scope a coordinating conjunction, and
5587 so on. These attachment decisions can be represented with a more lightweight structure:
5588 a directed graph over the words in the sentence, known as a **dependency parse**. Syntac-
5589 tic annotation has shifted its focus to such dependency structures: at the time of this
5590 writing, the **Universal Dependencies** project offers more than 100 dependency treebanks
5591 for more than 60 languages.¹ This chapter will describe the linguistic ideas underlying
5592 dependency grammar, and then discuss exact and transition-based parsing algorithms.
5593 The chapter will also discuss recent research on **learning to search** in transition-based
5594 structure prediction.

5595 11.1 Dependency grammar

5596 While **dependency grammar** has a rich history of its own (Tesnière, 1966; Kübler et al.,
5597 2009), it can be motivated by extension from the lexicalized context-free grammars that
5598 we encountered in previous chapter (§ 10.5.2). Recall that lexicalization augments each
5599 non-terminal with a **head word**. The head of a constituent is identified recursively, using
5600 a set of **head rules**, as shown in Table 10.3. An example of a lexicalized context-free parse
5601 is shown in Figure 11.1a. In this sentence, the head of the S constituent is the main verb,
5602 *scratch*; this non-terminal then produces the noun phrase *the cats*, whose head word is
5603 *cats*, and from which we finally derive the word *the*. Thus, the word *scratch* occupies the
5604 central position for the sentence, with the word *cats* playing a supporting role. In turn, *cats*

¹universaldependencies.org

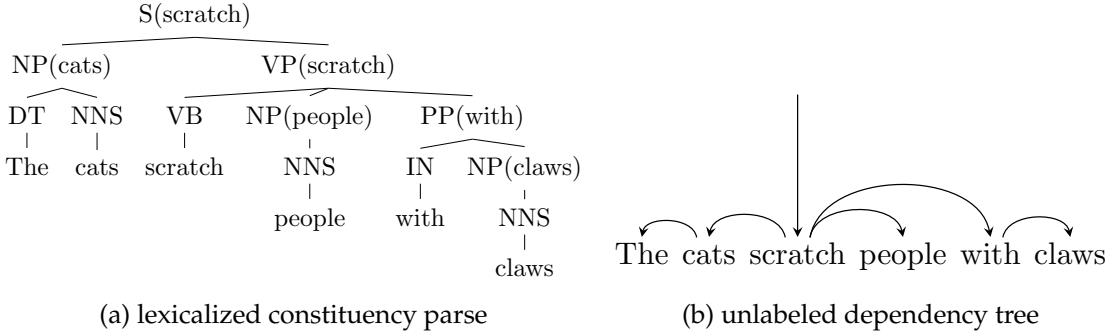


Figure 11.1: Dependency grammar is closely linked to lexicalized context free grammars: each lexical head has a dependency path to every other word in the constituent. (This example is based on the lexicalization rules from § 10.5.2, which make the preposition the head of a prepositional phrase. In the more contemporary Universal Dependencies annotations, the head of *with claws* would be *claws*, so there would be an edge *scratch* → *claws*.)

5605 occupies the central position for the noun phrase, with the word *the* playing a supporting
5606 role.

5607 The relationships between words in a sentence can be formalized in a directed graph,
5608 based on the lexicalized phrase-structure parse: create an edge (i, j) iff word i is the head
5609 of a phrase whose child is a phrase headed by word j . Thus, in our example, we would
5610 have *scratch* → *cats* and *cats* → *the*. We would not have the edge *scratch* → *the*, because
5611 although $S(\text{scratch})$ dominates $\text{DET}(\text{the})$ in the phrase-structure parse tree, it is not its im-
5612 mediate parent. These edges describe **syntactic dependencies**, a bilexical relationship
5613 between a **head** and a **dependent**, which is at the heart of dependency grammar.

5614 Continuing to build out this **dependency graph**, we will eventually reach every word
5615 in the sentence, as shown in Figure 11.1b. In this graph — and in all graphs constructed
5616 in this way — every word has exactly one incoming edge, except for the root word, which
5617 is indicated by a special incoming arrow from above. Furthermore, the graph is *weakly*
5618 *connected*: if the directed edges were replaced with undirected edges, there would be a
5619 path between all pairs of nodes. From these properties, it can be shown that there are no
5620 cycles in the graph (or else at least one node would have to have more than one incoming
5621 edge), and therefore, the graph is a tree. Because the graph includes all vertices, it is a
5622 **spanning tree**.

5623 11.1.1 Heads and dependents

5624 A dependency edge implies an asymmetric syntactic relationship between the head and
5625 dependent words, sometimes called **modifiers**. For a pair like *the cats* or *cats scratch*, how

5626 do we decide which is the head? Here are some possible criteria:

- 5627 • The head sets the syntactic category of the construction: for example, nouns are the
heads of noun phrases, and verbs are the heads of verb phrases.
- 5629 • The modifier may be optional while the head is mandatory: for example, in the
5630 sentence *cats scratch people with claws*, the subtrees *cats scratch* and *cats scratch people*
5631 are grammatical sentences, but *with claws* is not.
- 5632 • The head determines the morphological form of the modifier: for example, in lan-
5633 guages that require gender agreement, the gender of the noun determines the gen-
5634 der of the adjectives and determiners.
- 5635 • Edges should first connect content words, and then connect function words.

5636 As always, these guidelines sometimes conflict. The Universal Dependencies (UD)
5637 project has attempted to identify a set of principles that can be applied to dozens of dif-
5638 ferent languages (Nivre et al., 2016).² These guidelines are based on the universal part-
5639 of-speech tags from chapter 8. They differ somewhat from the head rules described in
5640 § 10.5.2: for example, on the principle that dependencies should relate content words, the
5641 prepositional phrase *with claws* would be headed by *claws*, resulting in an edge *scratch* →
5642 *claws*, and another edge *claws* → *with*.

5643 One objection to dependency grammar is that not all syntactic relations are asymmet-
5644 ric. Coordination is one of the most obvious examples (Popel et al., 2013): in the sentence,
5645 *Abigail and Max like kimchi* (Figure 11.2), which word is the head of the coordinated noun
5646 phrase *Abigail and Max*? Choosing either *Abigail* or *Max* seems arbitrary; fairness argues
5647 for making *and* the head, but this seems like the least important word in the noun phrase,
5648 and selecting it would violate the principle of linking content words first. The Universal
5649 Dependencies annotation system arbitrarily chooses the left-most item as the head — in
5650 this case, *Abigail* — and includes edges from this head to both *Max* and the coordinating
5651 conjunction *and*. These edges are distinguished by the labels CONJ (for the thing begin
5652 conjoined) and CC (for the coordinating conjunction). The labeling system is discussed
5653 next.

5654 11.1.2 Labeled dependencies

5655 Edges may be **labeled** to indicate the nature of the syntactic relation that holds between
5656 the two elements. For example, in Figure 11.2, the label NSUBJ on the edge from *like* to
5657 *Abigail* indicates that the subtree headed by *Abigail* is the noun subject of the verb *like*;
5658 similarly, the label OBJ on the edge from *like* to *kimchi* indicates that the subtree headed by

²The latest and most specific guidelines are available at universaldependencies.org/guidelines.html

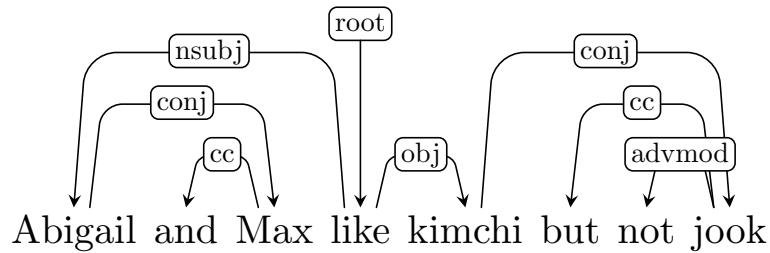


Figure 11.2: In the Universal Dependencies annotation system, the left-most item of a coordination is the head.

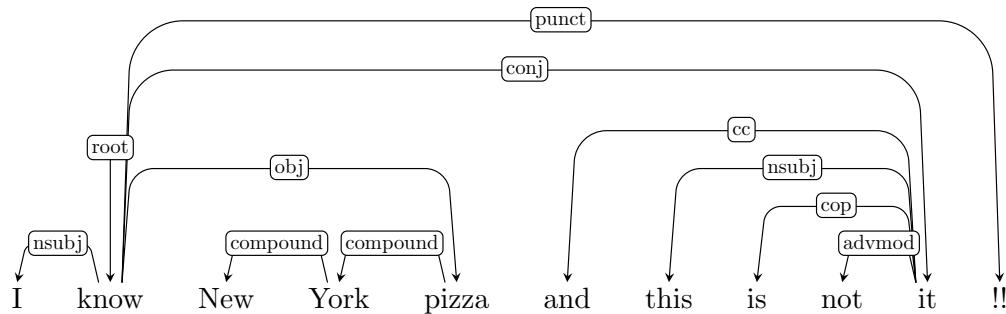


Figure 11.3: A labeled dependency parse from the English UD Treebank (reviews-361348-0006)

5659 *kimchi* is the object.³ The negation *not* is treated as an adverbial modifier (ADVMOD) on
5660 the noun *jook*.

5661 A slightly more complex example is shown in Figure 11.3. The multiword expression
5662 *New York pizza* is treated as a “flat” unit of text, with the elements linked by the COM-
5663 POUND relation. The sentence includes two clauses that are conjoined in the same way
5664 that noun phrases are conjoined in Figure 11.2. The second clause contains a **copula** verb
5665 (see § 8.1.1). For such clauses, we treat the “object” of the verb as the root — in this case,
5666 *it* — and label the verb as a dependent, with the COP relation. This example also shows
5667 how punctuations are treated, with label PUNCT.

5668 11.1.3 Dependency subtrees and constituents

5669 Dependency trees hide information that would be present in a CFG parse. Often what
5670 is hidden is in fact irrelevant: for example, Figure 11.4 shows three different ways of

³Earlier work distinguished direct and indirect objects (De Marneffe and Manning, 2008), but this has been dropped in version 2.0 of the Universal Dependencies annotation system.

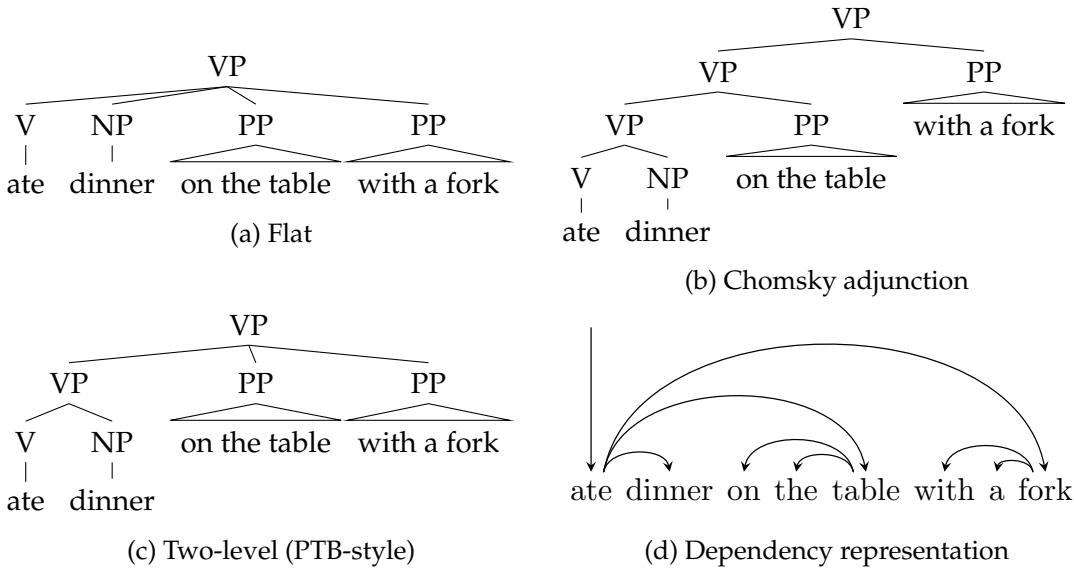


Figure 11.4: The three different CFG analyses of this verb phrase all correspond to a single dependency structure.

representing prepositional phrase adjuncts to the verb *ate*. Because there is apparently no meaningful difference between these analyses, the Penn Treebank decides by convention to use the two-level representation (see Johnson, 1998, for a discussion). As shown in Figure 11.4d, these three cases all look the same in a dependency parse.

But dependency grammar imposes its own set of annotation decisions, such as the identification of the head of a coordination (§ 11.1.1); without lexicalization, context-free grammar does not require either element in a coordination to be privileged in this way. Dependency parses can be disappointingly flat: for example, in the sentence *Yesterday, Abigail was reluctantly giving Max kimchi*, the root *giving* is the head of every dependency! The constituent parse arguably offers a more useful structural analysis for such cases.

Projectivity Thus far, we have defined dependency trees as spanning trees over a graph in which each word is a vertex. As we have seen, one way to construct such trees is by connecting the heads in a lexicalized constituent parse. However, there are spanning trees that cannot be constructed in this way. Syntactic constituents are *contiguous spans*. In a spanning tree constructed from a lexicalized constituent parse, the head h of any constituent that spans the nodes from i to j must have a path to every node in this span. This property is known as **projectivity**, and projective dependency parses are a restricted class of spanning trees. Informally, projectivity means that “crossing edges” are prohibited. The formal definition follows:

	% non-projective edges	% non-projective sentences
Czech	1.86%	22.42%
English	0.39%	7.63%
German	2.33%	28.19%

Table 11.1: Frequency of non-projective dependencies in three languages (Kuhlmann and Nivre, 2010)

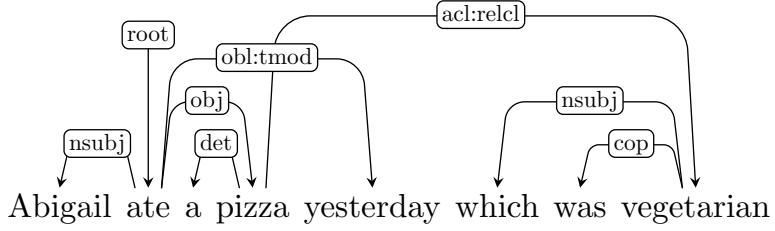


Figure 11.5: An example of a non-projective dependency parse. The “crossing edge” arises from the relative clause *which was vegetarian* and the oblique temporal modifier *yesterday*.

5690 **Definition 2** (Projectivity). *An edge from i to j is projective iff all k between i and j are descendants of i . A dependency parse is projective iff all its edges are projective.*

5692 Figure 11.5 gives an example of a non-projective dependency graph in English. This
 5693 dependency graph does not correspond to any constituent parse. As shown in Table 11.1,
 5694 non-projectivity is more common in languages such as Czech and German. Even though
 5695 relatively few dependencies are non-projective in these languages, many sentences have
 5696 at least one such dependency. As we will soon see, projectivity has important algorithmic
 5697 consequences.

5698 11.2 Graph-based dependency parsing

5699 Let $\mathbf{y} = \{i \xrightarrow{r} j\}$ represent a dependency graph, in which each edge is a relation r from
 5700 head word $i \in \{1, 2, \dots, M, \text{ROOT}\}$ to modifier $j \in \{1, 2, \dots, M\}$. The special node ROOT
 5701 indicates the root of the graph, and M is the length of the input $|\mathbf{w}|$. Given a scoring
 5702 function $\Psi(\mathbf{y}, \mathbf{w}; \theta)$, the optimal parse is,

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}(\mathbf{w})}{\operatorname{argmax}} \Psi(\mathbf{y}, \mathbf{w}; \theta), \quad [11.1]$$

5703 where $\mathcal{Y}(\mathbf{w})$ is the set of valid dependency parses on the input \mathbf{w} . As usual, the number
 5704 of possible labels $|\mathcal{Y}(\mathbf{w})|$ is exponential in the length of the input (Wu and Chao, 2004).

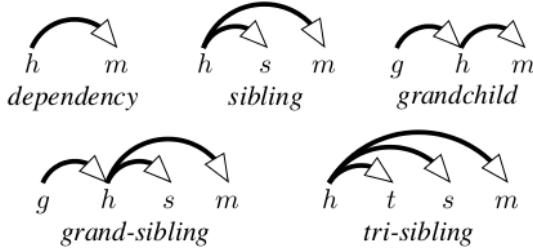


Figure 11.6: Feature templates for higher-order dependency parsing (Koo and Collins, 2010) [todo: permission]

5705 Algorithms that search over this space of possible graphs are known as **graph-based de-**
5706 **pendency parsers.**

In sequence labeling and constituent parsing, it was possible to search efficiently over an exponential space by choosing a feature function that decomposes into a sum of local feature vectors. A similar approach is possible for dependency parsing, by requiring the scoring function to decompose across dependency arcs $i \rightarrow j$:

$$\Psi(\mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}). \quad [11.2]$$

5707 Dependency parsers that operate under this assumption are known as **arc-factored**, since
5708 the overall score is a product of scores over all arcs.

Higher-order dependency parsing The arc-factored decomposition can be relaxed to allow higher-order dependencies. In **second-order dependency parsing**, the scoring function may include grandparents and siblings, as shown by the templates in Figure 11.6. The scoring function is,

$$\begin{aligned} \Psi(\mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) = & \sum_{i \xrightarrow{r} j \in \mathbf{y}} \sum_{k \xrightarrow{r'} i \in \mathbf{y}} \psi_{\text{grandparent}}(i \xrightarrow{r} j, k, r', \mathbf{w}; \boldsymbol{\theta}) \\ & \sum_{\substack{i \xrightarrow{r'} s \in \mathbf{y} \\ s \neq j}} \psi_{\text{sibling}}(i \xrightarrow{r} j, s, r', \mathbf{w}; \boldsymbol{\theta}). \end{aligned} \quad [11.3]$$

5709 The top line scores computes a scoring function that includes the grandparent k ; the
5710 bottom line computes a scoring function for each sibling s . For projective dependency
5711 graphs, there are efficient algorithms for second-order and third-order dependency pars-
5712 ing (Eisner, 1996; McDonald and Pereira, 2006; Koo and Collins, 2010); for non-projective
5713 dependency graphs, second-order dependency parsing is NP-hard (McDonald and Pereira,
5714 2006). The specific algorithms are discussed in the next section.

5715 **11.2.1 Graph-based parsing algorithms**

5716 The distinction between projective and non-projective dependency trees (§ 11.1.3) plays
 5717 a key role in the choice of algorithms. Because projective dependency trees are closely
 5718 related to (and can be derived from) lexicalized constituent trees, lexicalized parsing al-
 5719 gorithms can be applied directly. For the more general problem of parsing to arbitrary
 5720 spanning trees, a different class of algorithms is required. In both cases, arc-factored de-
 5721 pendency parsing relies on precomputing the scores $\psi(i \xrightarrow{r} j, w; \theta)$ for each potential
 5722 edge. There are $\mathcal{O}(M^2 R)$ such scores, where M is the length of the input and R is the
 5723 number of dependency relation types, and this is a lower bound on the time and space
 5724 complexity of any exact algorithm for arc-factored dependency parsing.

5725 **11.2.1.1 Projective dependency parsing**

5726 Any lexicalized constituency tree can be converted into a projective dependency tree by
 5727 creating arcs between the heads of constituents and their parents, so any algorithm for
 5728 lexicalized constituent parsing can be converted into an algorithm for projective depen-
 5729 dency parsing, by converting arc scores into scores for lexicalized productions. As noted
 5730 in § 10.5.2, there are cubic time algorithms for lexicalized constituent parsing, which are
 5731 extensions of the CKY algorithm. Therefore, arc-factored projective dependency parsing
 5732 can be performed in cubic time in the length of the input.

5733 Second-order projective dependency parsing can also be performed in cubic time, with
 5734 minimal modifications to the lexicalized parsing algorithm (Eisner, 1996). It is possible to
 5735 go even further, to **third-order dependency parsing**, in which the scoring function may
 5736 consider great-grandparents, grand-siblings, and “tri-siblings”, as shown in Figure 11.6.
 5737 Third-order dependency parsing can be performed in $\mathcal{O}(M^4)$ time, which can be made
 5738 practical through the use of pruning to eliminate unlikely edges (Koo and Collins, 2010).

5739 **11.2.1.2 Non-projective dependency parsing**

5740 In non-projective dependency parsing, the goal is to identify the highest-scoring span-
 5741 ning tree over the words in the sentence. The arc-factored assumption ensures that the
 5742 score for each spanning tree will be computed as a sum over scores for the edges, which
 5743 are precomputed. Based on these scores, we build a weighted connected graph. Arc-
 5744 factored non-projective dependency parsing is then equivalent to finding the spanning
 5745 tree that achieves the maximum total score, $\Psi(y, w) = \sum_{i \xrightarrow{r} j \in y} \psi(i \xrightarrow{r} j, w)$. The **Chu-**
 5746 **Liu-Edmonds algorithm** (Chu and Liu, 1965; Edmonds, 1967) computes this **maximum**
 5747 **spanning tree** efficiently. It does this by first identifying the best incoming edge $i \xrightarrow{r} j$ for
 5748 each vertex j . If the resulting graph does not contain cycles, it is the maximum spanning
 5749 tree. If there is a cycle, it is collapsed into a super-vertex, whose incoming and outgoing
 5750 edges are based on the edges to the vertices in the cycle. The algorithm is then applied

5751 recursively to the resulting graph, and process repeats until a graph without cycles is
 5752 obtained.

5753 The time complexity of identifying the best incoming edge for each vertex is $\mathcal{O}(M^2R)$,
 5754 where M is the length of the input and R is the number of relations; in the worst case, the
 5755 number of cycles is $\mathcal{O}(M)$. Therefore, the complexity of the Chu-Liu-Edmonds algorithm
 5756 is $\mathcal{O}(M^3R)$. This complexity can be reduced to $\mathcal{O}(M^2N)$ by storing the edge scores in a
 5757 Fibonacci heap (Gabow et al., 1986). For more detail on graph-based parsing algorithms,
 5758 see Eisner (1997) and Kübler et al. (2009).

5759 **Higher-order non-projective dependency parsing** Given the tractability of higher-order
 5760 projective dependency parsing, you may be surprised to learn that non-projective second-
 5761 order dependency parsing is NP-Hard. This can be proved by reduction from the vertex
 5762 cover problem (Neuhaus and Bröker, 1997). A heuristic solution is to do projective pars-
 5763 ing first, and then post-process the projective dependency parse to add non-projective
 5764 edges (Nivre and Nilsson, 2005). More recent work has applied techniques for approxi-
 5765 mate inference in graphical models, including belief propagation (Smith and Eisner, 2008),
 5766 integer linear programming (Martins et al., 2009), variational inference (Martins et al.,
 5767 2010), and Markov Chain Monte Carlo (Zhang et al., 2014).

5768 11.2.2 Computing scores for dependency arcs

The arc-factored scoring function $\psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})$ can be defined in several ways:

$$\text{Linear} \quad \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \boldsymbol{\theta} \cdot \mathbf{f}(i \xrightarrow{r} j, \mathbf{w}) \quad [11.4]$$

$$\text{Neural} \quad \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \text{Feedforward}([\mathbf{u}_{w_i}; \mathbf{u}_{w_j}]; \boldsymbol{\theta}) \quad [11.5]$$

$$\text{Generative} \quad \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \log p(w_j, r | w_i). \quad [11.6]$$

5769 11.2.2.1 Linear feature-based arc scores

5770 Linear models for dependency parsing incorporate many of the same features used in
 5771 sequence labeling and discriminative constituent parsing. These include:

- 5772 • the length and direction of the arc;
- 5773 • the words w_i and w_j linked by the dependency relation;
- 5774 • the prefixes, suffixes, and parts-of-speech of these words;
- 5775 • the neighbors of the dependency arc, $w_{i-1}, w_{i+1}, w_{j-1}, w_{j+1}$;
- 5776 • the prefixes, suffixes, and part-of-speech of these neighbor words.

5777 Each of these features can be conjoined with the dependency edge label r . Note that
 5778 features in an arc-factored parser can refer to words other than w_i and w_j . The restriction
 5779 is that the features consider only a single arc.

Bilexical features (e.g., *sushi* → *chopsticks*) are powerful but rare, so it is useful to augment them with coarse-grained alternatives, by “backing off” to the part-of-speech or affix. For example, the following features are created by backing off to part-of-speech tags in an unlabeled dependency parser:

$$\begin{aligned} f(3 \rightarrow 5, \text{we eat sushi with chopsticks}) = & \langle \text{sushi} \rightarrow \text{chopsticks}, \\ & \text{sushi} \rightarrow \text{NNS}, \\ & \text{NN} \rightarrow \text{chopsticks}, \\ & \text{NNS} \rightarrow \text{NN} \rangle. \end{aligned}$$

5780 Regularized discriminative learning algorithms can then trade off between features at
 5781 varying levels of detail. McDonald et al. (2005) take this approach as far as *tetralexical*
 5782 features (e.g., $(w_i, w_{i+1}, w_{j-1}, w_j)$). Such features help to avoid choosing arcs that are un-
 5783 likely due to the intervening words: for example, there is unlikely to be an edge between
 5784 two nouns if the intervening span contains a verb. A large list of first and second-order
 5785 features is provided by Bohnet (2010), who uses a hashing function to store these features
 5786 efficiently.

5787 11.2.2.2 Neural arc scores

Given vector representations \mathbf{x}_i for each word w_i in the input, a set of arc scores can be computed from a feedforward neural network:

$$\psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \text{FeedForward}([\mathbf{x}_i; \mathbf{x}_j]; \boldsymbol{\theta}_r), \quad [11.7]$$

where unique weights $\boldsymbol{\theta}_r$ are available for each arc type (Pei et al., 2015; Kiperwasser and Goldberg, 2016). Kiperwasser and Goldberg (2016) use a feedforward network with a single hidden layer,

$$\mathbf{z} = g(\boldsymbol{\Theta}_r[\mathbf{x}_i; \mathbf{x}_j] + \mathbf{b}_r^{(z)}) \quad [11.8]$$

$$\psi(i \xrightarrow{r} j) = \boldsymbol{\beta}_r \mathbf{z} + \mathbf{b}_r^{(y)}, \quad [11.9]$$

5788 where $\boldsymbol{\Theta}_r$ is a matrix, $\boldsymbol{\beta}_r$ is a vector, each \mathbf{b}_r is a scalar, and the function g is an elementwise
 5789 \tanh activation function.

5790 The vector \mathbf{x}_i can be set equal to the word embedding, which may be pre-trained or
 5791 learned by backpropagation (Pei et al., 2015). Alternatively, contextual information can
 5792 be incorporated by applying a bidirectional recurrent neural network across the input, as
 5793 described in § 7.6. The RNN hidden states at each word can be used as inputs to the arc
 5794 scoring function (Kiperwasser and Goldberg, 2016).

5795 **11.2.2.3 Probabilistic arc scores**

If each arc score is equal to the log probability $\log p(w_j, r \mid w_i)$, then the sum of scores gives the log probability of the sentence and arc labels, by the chain rule. For example, consider the unlabeled parse of *we eat sushi with rice*,

$$\mathbf{y} = \{(ROOT, 2), (2, 1), (2, 3), (3, 5), (5, 4)\} \quad [11.10]$$

$$\log p(\mathbf{w} \mid \mathbf{y}) = \sum_{(i \rightarrow j) \in \mathbf{y}} \log p(w_j \mid w_i) \quad [11.11]$$

$$\begin{aligned} &= \log p(eat \mid ROOT) + \log p(we \mid eat) + \log p(sushi \mid eat) \\ &\quad + \log p(rice \mid sushi) + \log p(with \mid rice). \end{aligned} \quad [11.12]$$

5796 Probabilistic generative models are used in combination with expectation-maximization
 5797 (chapter 5) for unsupervised dependency parsing (Klein and Manning, 2004).

5798 **11.2.3 Learning**

Having formulated graph-based dependency parsing as a structure prediction problem, we can apply similar learning algorithms to those used in sequence labeling. Given a loss function $\ell(\boldsymbol{\theta}; \mathbf{w}^{(i)}, \mathbf{y}^{(i)})$, we can compute gradient-based updates to the parameters. For a model with feature-based arc scores and a perceptron loss, we obtain the usual structured perceptron update,

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \mathbf{y}') \quad [11.13]$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \mathbf{f}(\mathbf{w}, \mathbf{y}) - \mathbf{f}(\mathbf{w}, \hat{\mathbf{y}}) \quad [11.14]$$

5799 In this case, the argmax requires a maximization over all dependency trees for the sen-
 5800 tence, which can be computed using the algorithms described in § 11.2.1. We can apply
 5801 all the usual tricks from § 2.2: weight averaging, a large margin objective, and regular-
 5802 ization. McDonald et al. (2005) were the first to treat dependency parsing as a structure
 5803 prediction problem, using MIRA, an online margin-based learning algorithm. Neural arc
 5804 scores can be learned in the same way, backpropagating from a margin loss to updates on
 5805 the feedforward network that computes the score for each edge.

A conditional random field for arc-factored dependency parsing is built on the probability model,

$$p(\mathbf{y} \mid \mathbf{w}) = \frac{\exp \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})}{\sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp \sum_{i \xrightarrow{r} j \in \mathbf{y}'} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})} \quad [11.15]$$

5806 Such a model is trained to minimize the negative log conditional-likelihood. Just as in
 5807 CRF sequence models (§ 7.5.3) and the logistic regression classifier (§ 2.4), the gradients

5808 involve marginal probabilities $p(i \xrightarrow{r} j \mid \mathbf{w}; \theta)$, which in this case are probabilities over
 5809 individual dependencies. In arc-factored models, these probabilities can be computed
 5810 in polynomial time. For projective dependency trees, the marginal probabilities can be
 5811 computed in cubic time, using a variant of the inside-outside algorithm (Lari and Young,
 5812 1990). For non-projective dependency parsing, marginals can also be computed in cubic
 5813 time, using the **matrix-tree theorem** (Koo et al., 2007; McDonald et al., 2007; Smith and
 5814 Smith, 2007). Details of these methods are described by Kübler et al. (2009).

5815 11.3 Transition-based dependency parsing

5816 Graph-based dependency parsing offers exact inference, meaning that it is possible to re-
 5817 cover the best-scoring parse for any given model. But this comes at a price: the scoring
 5818 function is required to decompose into local parts — in the case of non-projective parsing,
 5819 these parts are restricted to individual arcs. These limitations are felt more keenly in de-
 5820 pendency parsing than in sequence labeling, because second-order dependency features
 5821 are critical to correctly identify some types of attachments. For example, prepositional
 5822 phrase attachment depends on the attachment point, the object of the preposition, and
 5823 the preposition itself; arc-factored scores cannot account for all three of these features si-
 5824 multaneously. Graph-based dependency parsing may also be criticized on the basis of
 5825 intuitions about human language processing: people read and listen to sentences *sequen-*
 5826 *tially*, incrementally building mental models of the sentence structure and meaning before
 5827 getting to the end (Jurafsky, 1996). This seems hard to reconcile with graph-based algo-
 5828 rithms, which perform bottom-up operations on the entire sentence, requiring the parser
 5829 to keep every word in memory. Finally, from a practical perspective, graph-based depen-
 5830 dency parsing is relatively slow, running in cubic time in the length of the input.

5831 Transition-based algorithms address all three of these objections. They work by mov-
 5832 ing through the sentence sequentially, while performing actions that incrementally up-
 5833 date a stored representation of what has been read thus far. As with the shift-reduce
 5834 parser from § 10.6.2, this representation consists of a stack, onto which parsing substruc-
 5835 tures can be pushed and popped. In shift-reduce, these substructures were constituents;
 5836 in the transition systems that follow, they will be projective dependency trees over partial
 5837 spans of the input.⁴ Parsing is complete when the input is consumed and there is only
 5838 a single structure on the stack. The sequence of actions that led to the parse is known as
 5839 the **derivation**. One problem with transition-based systems is that there may be multiple
 5840 derivations for a single parse structure — a phenomenon known as **spurious ambiguity**.

⁴Transition systems also exist for non-projective dependency parsing (e.g., Nivre, 2008).

5841 **11.3.1 Transition systems for dependency parsing**

5842 A **transition system** consists of a representation for describing configurations of the parser,
 5843 and a set of transition actions, which manipulate the configuration. There are two main
 5844 transition systems for dependency parsing: **arc-standard**, which is closely related to shift-
 5845 reduce, and **arc-eager**, which adds an additional action that can simplify derivations (Ab-
 5846 ney and Johnson, 1991). In both cases, transitions are between **configurations** that are
 5847 represented as triples, $C = (\sigma, \beta, A)$, where σ is the stack, β is the input buffer, and A is
 5848 the list of arcs that have been created (Nivre, 2008). In the initial configuration,

$$C_{\text{initial}} = ([\text{ROOT}], \mathbf{w}, \emptyset), \quad [11.16]$$

5849 indicating that the stack contains only the special node ROOT, the entire input is on the
 5850 buffer, and the set of arcs is empty. An accepting configuration is,

$$C_{\text{accept}} = ([\text{ROOT}], \emptyset, A), \quad [11.17]$$

5851 where the stack contains only ROOT, the buffer is empty, and the arcs A define a spanning
 5852 tree over the input. The arc-standard and arc-eager systems define a set of transitions
 5853 between configurations, which are capable of transforming an initial configuration into
 5854 an accepting configuration. In both of these systems, the number of actions required to
 5855 parse an input grows linearly in the length of the input, making transition-based parsing
 5856 considerably more efficient than graph-based methods.

5857 **11.3.1.1 Arc-standard**

5858 The **arc-standard** transition system is closely related to shift-reduce, and to the LR algo-
 5859 rithm that is used to parse programming languages (Aho et al., 2006). It includes the
 5860 following classes of actions:

- 5861 • SHIFT: move the first item from the input buffer on to the top of the stack,

$$(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A), \quad [11.18]$$

5862 where we write $i|\beta$ to indicate that i is the leftmost item in the input buffer, and $\sigma|i$
 5863 to indicate the result of pushing i on to stack σ .

- 5864 • ARC-LEFT: create a new left-facing arc of type r between the item on the top of the
 5865 stack and the first item in the input buffer. The head of this arc is j , which remains
 5866 at the front of the input buffer. The arc $j \xrightarrow{r} i$ is added to A . Formally,

$$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \oplus j \xrightarrow{r} i), \quad [11.19]$$

5867 where r is the label of the dependency arc, and \oplus concatenates the new arc $j \xrightarrow{r} i$ to
 5868 the list A .

σ	β	action	arc added to \mathcal{A}
1. [ROOT]	<i>they like bagels with lox</i>	SHIFT	
2. [ROOT, <i>they</i>]	<i>like bagels with lox</i>	ARC-LEFT	(<i>they</i> \leftarrow <i>like</i>)
3. [ROOT]	<i>like bagels with lox</i>	SHIFT	
4. [ROOT, <i>like</i>]	<i>bagels with lox</i>	SHIFT	
5. [ROOT, <i>like</i> , <i>bagels</i>]	<i>with lox</i>	SHIFT	
6. [ROOT, <i>like</i> , <i>bagels</i> , <i>with</i>]	<i>lox</i>	ARC-LEFT	(<i>with</i> \leftarrow <i>lox</i>)
7. [ROOT, <i>like</i> , <i>bagels</i>]	<i>lox</i>	ARC-RIGHT	(<i>bagels</i> \rightarrow <i>lox</i>)
8. [ROOT, <i>like</i>]	<i>bagels</i>	ARC-RIGHT	(<i>like</i> \rightarrow <i>bagels</i>)
9. [ROOT]	<i>like</i>	ARC-RIGHT	(ROOT \rightarrow <i>like</i>)
10. [ROOT]	\emptyset	DONE	

Table 11.2: Arc-standard derivation of the unlabeled dependency parse for the input *they like bagels with lox*.

- 5869 • ARC-RIGHT: creates a new right-facing arc of type r between the item on the top of
 5870 the stack and the first item in the input buffer. The head of this arc is i , which is
 5871 “popped” from the stack and pushed to the front of the input buffer. The arc $i \xrightarrow{r} j$
 5872 is added to A . Formally,

$$(\sigma | i, j | \beta, A) \Rightarrow (\sigma, i | \beta, A \oplus i \xrightarrow{r} j), \quad [11.20]$$

5873 where again r is the label of the dependency arc.

5874 Each action has preconditions. The SHIFT action can be performed only when the buffer
 5875 has at least one element. The ARC-LEFT action cannot be performed when the root node
 5876 ROOT is on top of the stack, since this node must be the root of the entire tree. The ARC-
 5877 LEFT and ARC-RIGHT remove the modifier words from the stack (in the case of ARC-LEFT)
 5878 and from the buffer (in the case of ARC-RIGHT), so it is impossible for any word to have
 5879 more than one parent. Furthermore, the end state can only be reached when every word is
 5880 removed from the buffer and stack, so the set of arcs is guaranteed to constitute a spanning
 5881 tree. An example arc-standard derivation is shown in Table 11.2.

5882 11.3.1.2 Arc-eager dependency parsing

5883 In the arc-standard transition system, a word is completely removed from the parse once
 5884 it has been made the modifier in a dependency arc. At this time, any dependents of
 5885 this word must have already been identified. Right-branching structures are common in
 5886 English (and many other languages), with words often modified by units such as prepo-
 5887 sitional phrases to their right. In the arc-standard system, this means that we must first
 5888 shift all the units of the input onto the stack, and then work backwards, creating a series of

5889 arcs, as occurs in Table 11.2. Note that the decision to shift *bagels* onto the stack guarantees
 5890 that the prepositional phrase *with lox* will attach to the noun phrase, and that this decision
 5891 must be made before the prepositional phrase is itself parsed. This has been argued to be
 5892 cognitively implausible (Abney and Johnson, 1991); from a computational perspective, it
 5893 means that a parser may need to look several steps ahead to make the correct decision.

5894 **Arc-eager dependency parsing** changes the ARC-RIGHT action so that right depen-
 5895 dents can be attached before all of their dependents have been found. Rather than re-
 5896 moving the modifier from both the buffer and stack, the ARC-RIGHT action pushes the
 5897 modifier on to the stack, on top of the head. Because the stack can now contain elements
 5898 that already have parents in the partial dependency graph, two additional changes are
 5899 necessary:

- 5900 • A precondition is required to ensure that the ARC-LEFT action cannot be applied
 5901 when the top element on the stack already has a parent in A .
- 5902 • A new REDUCE action is introduced, which can remove elements from the stack if
 5903 they already have a parent in A :

$$(\sigma|i, \beta, A) \Rightarrow (\sigma, \beta, A). \quad [11.21]$$

5904 As a result of these changes, it is now possible to create the arc *like* \rightarrow *bagels* before parsing
 5905 the prepositional phrase *with lox*. Furthermore, this action does not imply a decision about
 5906 whether the prepositional phrase will attach to the noun or verb. Noun attachment is
 5907 chosen in the parse in Table 11.3, but verb attachment could be achieved by applying the
 5908 REDUCE action at step 5 or 7.

5909 11.3.1.3 Projectivity

5910 The arc-standard and arc-eager transition systems are guaranteed to produce projective
 5911 dependency trees, because all arcs are between the word at the top of the stack and the
 5912 left-most edge of the buffer (Nivre, 2008). Non-projective transition systems can be con-
 5913 structed by adding actions that create arcs with words that are second or third in the
 5914 stack (Attardi, 2006), or by adopting an alternative configuration structure, which main-
 5915 tains a list of all words that do not yet have heads (Covington, 2001). In **pseudo-projective**
 5916 **dependency parsing**, a projective dependency parse is generated first, and then a set of
 5917 graph transformation techniques are applied, producing non-projective edges (Nivre and
 5918 Nilsson, 2005).

5919 11.3.1.4 Beam search

5920 In “greedy” transition-based parsing, the parser tries to make the best decision at each
 5921 configuration. This can lead to search errors, when an early decision locks the parser into

σ	β	action	arc added to \mathcal{A}
1. [ROOT]	<i>they like bagels with lox</i>	SHIFT	
2. [ROOT, <i>they</i>]	<i>like bagels with lox</i>	ARC-LEFT	(<i>they</i> \leftarrow <i>like</i>)
3. [ROOT]	<i>like bagels with lox</i>	ARC-RIGHT	(ROOT \rightarrow <i>like</i>)
4. [ROOT, <i>like</i>]	<i>bagels with lox</i>	ARC-RIGHT	(<i>like</i> \rightarrow <i>bagels</i>)
5. [ROOT, <i>like</i> , <i>bagels</i>]	<i>with lox</i>	SHIFT	
6. [ROOT, <i>like</i> , <i>bagels</i> , <i>with</i>]	<i>lox</i>	ARC-LEFT	(<i>with</i> \leftarrow <i>lox</i>)
7. [ROOT, <i>like</i> , <i>bagels</i>]	<i>lox</i>	ARC-RIGHT	(<i>bagels</i> \rightarrow <i>lox</i>)
8. [ROOT, <i>like</i> , <i>bagels</i> , <i>lox</i>]	\emptyset	REDUCE	
9. [ROOT, <i>like</i> , <i>bagels</i>]	\emptyset	REDUCE	
10. [ROOT, <i>like</i>]	\emptyset	REDUCE	
11. [ROOT]	\emptyset	DONE	

Table 11.3: Arc-eager derivation of the unlabeled dependency parse for the input *they like bagels with lox*.

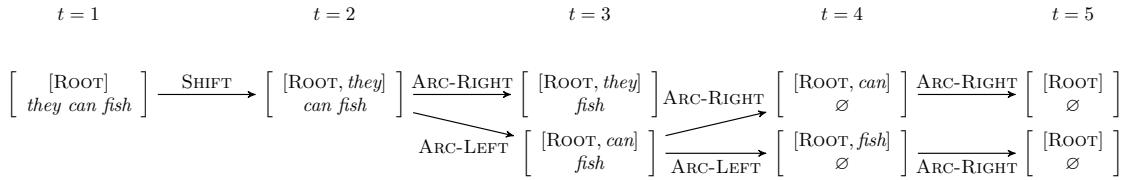


Figure 11.7: Beam search for unlabeled dependency parsing, with beam size $K = 2$. The arc lists for each configuration are not shown, but can be computed from the transitions.

5922 a poor derivation. For example, in Table 11.2, if ARC-RIGHT were chosen at step 4, then
 5923 the parser would later be forced to attach the prepositional phrase *with lox* to the verb
 5924 *likes*. Note that the *likes* \rightarrow *bagels* arc is indeed part of the correct dependency parse, but
 5925 the arc-standard transition system requires it to be created later in the derivation.

Beam search addresses this issue by maintaining a set of hypothetical derivations, called a beam. At step t of the derivation, there is a set of k hypotheses, each of which is a tuple of a score and a sequence of actions,

$$h_t^{(k)} = (s_t^{(k)}, A_t^{(k)}) \quad [11.22]$$

5926 Each hypothesis is then “expanded” by considering the set of all valid actions from the
 5927 current configuration $c_t^{(k)}$, written $\mathcal{A}(c_t^{(k)})$. This yields a large set of new hypotheses. For
 5928 each action $a \mathcal{A}(c_t^{(k)})$, we score the new hypothesis $A_t^{(k)} \oplus a$. The top k hypotheses by
 5929 this scoring metric are kept, and parsing proceeds to the next step (Zhang and Clark,

5930 2008). Note that beam search requires a scoring function for action *sequences*, rather than
 5931 individual actions. This issue will be revisited in the next section.

5932 An example of beam search is shown in Figure 11.7, with a beam size of $K = 2$. For the
 5933 first transition, the only valid action is SHIFT, so there is only one possible configuration
 5934 at $t = 2$. From this configuration, there are three possible actions. The top two are ARC-
 5935 RIGHT and ARC-LEFT, and so the resulting hypotheses from these actions are on the beam
 5936 at $t = 3$. From these configurations, there are three possible actions each, but the best
 5937 two are expansions of the bottom hypothesis at $t = 3$. Parsing continues until $t = 5$, at
 5938 which point both hypotheses reach an accepting state. The best-scoring hypothesis is then
 5939 selected as the parse.

5940 11.3.2 Scoring functions for transition-based parsers

Transition-based parsing requires selecting a series of actions. In greedy transition-based
 parsing, this can be done by training a classifier,

$$\hat{a} = \underset{a \in \mathcal{A}(c)}{\operatorname{argmax}} \Psi(a, c, \mathbf{w}; \boldsymbol{\theta}), \quad [11.23]$$

5941 where $\mathcal{A}(c)$ is the set of admissible actions in the current configuration c , \mathbf{w} is the input,
 5942 and Ψ is a scoring function with parameters $\boldsymbol{\theta}$ (Yamada and Matsumoto, 2003).

5943 A feature-based score can be computed, $\Psi(a, c, \mathbf{w}) = \boldsymbol{\theta} \cdot \mathbf{f}(a, c, \mathbf{w})$, using features that
 5944 may consider any aspect of the current configuration and input sequence. Typical features
 5945 for transition-based dependency parsing include: the word and part-of-speech of the top
 5946 element on the stack; the word and part-of-speech of the first, second, and third elements
 5947 on the input buffer; pairs and triples of words and parts-of-speech from the top of the
 5948 stack and the front of the buffer; the distance (in tokens) between the element on the top
 5949 of the stack and the element in the front of the input buffer; the number of modifiers of
 5950 each of these elements; and higher-order dependency features as described above in the
 5951 section on graph-based dependency parsing (see, e.g., Zhang and Nivre, 2011).

5952 Parse actions can also be scored by neural networks. For example, Chen and Manning
 5953 (2014) build a feedforward network in which the input layer consists of the concatenation
 5954 of embeddings of several words and tags:

- 5955 • the top three words on the stack, and the first three words on the buffer;
- 5956 • the first and second leftmost and rightmost children (dependents) of the top two
 5957 words on the stack;
- 5958 • the leftmost and right most grandchildren of the top two words on the stack;
- 5959 • embeddings of the part-of-speech tags of these words.

Let us call this base layer $\mathbf{x}(c, \mathbf{w})$, defined as,

$$c = (\sigma, \beta, A)$$

$$\mathbf{x}(c, \mathbf{w}) = [\mathbf{v}_{w_{\sigma_1}}, \mathbf{v}_{t_{\sigma_1}} \mathbf{v}_{w_{\sigma_2}}, \mathbf{v}_{t_{\sigma_2}}, \mathbf{v}_{w_{\sigma_3}}, \mathbf{v}_{t_{\sigma_3}}, \mathbf{v}_{w_{\beta_1}}, \mathbf{v}_{t_{\beta_1}}, \mathbf{v}_{w_{\beta_2}}, \mathbf{v}_{t_{\beta_2}}, \dots],$$

where $\mathbf{v}_{w_{\sigma_1}}$ is the embedding of the first word on the stack, $\mathbf{v}_{t_{\beta_2}}$ is the embedding of the part-of-speech tag of the second word on the buffer, and so on. Given this base encoding of the parser state, the score for the set of possible actions is computed through a feedforward network,

$$\mathbf{z} = g(\Theta^{(x \rightarrow z)} \mathbf{x}(c, \mathbf{w})) \quad [11.24]$$

$$\psi(a, c, \mathbf{w}; \boldsymbol{\theta}) = \Theta_a^{(z \rightarrow y)} \mathbf{z}, \quad [11.25]$$

5960 where the vector \mathbf{z} plays the same role as the features $\mathbf{f}(a, c, \mathbf{w})$, but is a learned representation.
 5961 Chen and Manning (2014) use a cubic elementwise activation function, $g(x) = x^3$,
 5962 so that the hidden layer models products across all triples of input features. The learning
 5963 algorithm updates the embeddings as well as the parameters of the feedforward network.

5964 11.3.3 Learning to parse

5965 Transition-based dependency parsing suffers from a mismatch between the supervision,
 5966 which comes in the form of dependency trees, and the classifier's prediction space, which
 5967 is a set of parsing actions. One solution is to create new training data by converting parse
 5968 trees into action sequences; another is to derive supervision directly from the parser's
 5969 performance.

5970 11.3.3.1 Oracle-based training

5971 A transition system can be viewed as a function from action sequences (also called **derivations**)
 5972 to parse trees. The inverse of this function is a mapping from parse trees to derivations,
 5973 which is called an **oracle**. For the arc-standard and arc-eager parsing system, an
 5974 oracle can be computed in linear time in the length of the derivation (Kübler et al., 2009,
 5975 page 32). Both the arc-standard and arc-eager transition systems suffer from **spurious**
 5976 **ambiguity**: there exist dependency parses for which multiple derivations are possible,
 5977 such as $1 \leftarrow 2 \rightarrow 3$. The oracle must choose between these different derivations. For exam-
 5978 ple, the algorithm described by Kübler et al. (2009) would first create the left arc ($1 \leftarrow 2$),
 5979 and then create the right arc, $(1 \leftarrow 2) \rightarrow 3$; another oracle might begin by shifting twice,
 5980 resulting in the derivation $1 \leftarrow (2 \rightarrow 3)$.

Given such an oracle, a dependency treebank can be converted into a set of oracle action sequences $\{A^{(i)}\}_{i=1}^N$. The parser can be trained by stepping through the oracle action sequences, and optimizing on an classification-based objective that rewards selecting the

oracle action. For transition-based dependency parsing, maximum conditional likelihood is a typical choice (Chen and Manning, 2014; Dyer et al., 2015):

$$p(a | c, \mathbf{w}) = \frac{\exp \Psi(a, c, \mathbf{w}; \boldsymbol{\theta})}{\sum_{a' \in \mathcal{A}(c)} \exp \Psi(a', c, \mathbf{w}; \boldsymbol{\theta})} \quad [11.26]$$

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^N \sum_{t=1}^{|A^{(i)}|} \log p(a_t^{(i)} | c_t^{(i)}, \mathbf{w}), \quad [11.27]$$

5981 where $|A^{(i)}|$ is the length of the action sequence $A^{(i)}$.

5982 Recall that beam search requires a scoring function for action sequences. Such a score
 5983 can be obtained by adding the log-likelihoods (or hinge losses) across all actions in the
 5984 sequence (Chen and Manning, 2014).

5985 11.3.3.2 Global objectives

5986 The objective in Equation 11.27 is **locally-normalized**: it is the product of normalized
 5987 probabilities over individual actions. A similar characterization could be made of non-
 5988 probabilistic algorithms in which hinge-loss objectives are summed over individual ac-
 5989 tions. In either case, training on individual actions can be sub-optimal with respect to
 5990 global performance, due to the **label bias problem** (Lafferty et al., 2001; Andor et al.,
 5991 2016).

5992 As a stylized example, suppose that a given configuration appears 100 times in the
 5993 training data, with action a_1 as the oracle action in 51 cases, and a_2 as the oracle action in
 5994 the other 49 cases. However, in cases where a_2 is correct, choosing a_1 results in a cascade
 5995 of subsequent errors, while in cases where a_1 is correct, choosing a_2 results in only a single
 5996 error. A classifier that is trained on a local objective function will learn to always choose
 5997 a_1 , but choosing a_2 would minimize the overall number of errors.

5998 This observation motivates a global objective, such as the globally-normalized condi-
 5999 tional likelihood,

$$p(A^{(i)} | \mathbf{w}; \boldsymbol{\theta}) = \frac{\exp \sum_{t=1}^{|A^{(i)}|} \Psi(a_t^{(i)}, c_t^{(i)}, \mathbf{w})}{\sum_{A' \in \mathbb{A}(\mathbf{w})} \exp \sum_{t=1}^{|A'|} \Psi(a'_t, c'_t, \mathbf{w})}, \quad [11.28]$$

where the denominator sums over the set of all possible action sequences, $\mathbb{A}(\mathbf{w})$.⁵ In the conditional random field model for sequence labeling (§ 7.5.3), it was possible to compute

⁵Andor et al. (2016) prove that the set of globally-normalized conditional distributions is a strict superset of the set of locally-normalized conditional distributions, and that globally-normalized conditional models are therefore strictly more expressive.

this sum explicitly, using dynamic programming. In transition-based parsing, this is not possible. However, the sum can be approximated using beam search,

$$\sum_{A' \in \mathbb{A}(\mathbf{w})} \exp \sum_{t=1}^{|A'|} \Psi(a'_t, c'_t, \mathbf{w}) \approx \sum_{k=1}^K \exp \sum_{t=1}^{|A^{(k)}|} \Psi(a_t^{(k)}, c_t^{(k)}, \mathbf{w}), \quad [11.29]$$

where $A^{(k)}$ is an action sequence on a beam of size K . This gives rise to the following loss function,

$$L(\boldsymbol{\theta}) = - \sum_{t=1}^{|A^{(i)}|} \Psi(a_t^{(i)}, c_t^{(i)}, \mathbf{w}) + \log \sum_{k=1}^K \exp \sum_{t=1}^{|A^{(k)}|} \Psi(a_t^{(k)}, c_t^{(k)}, \mathbf{w}). \quad [11.30]$$

The derivatives of this loss involve expectations with respect to a probability distribution over action sequences on the beam.

11.3.3.3 *Early update and the incremental perceptron

When learning in the context of beam search, the goal is to learn a decision function so that the gold dependency parse is always reachable from at least one of the partial derivations on the beam. (The combination of a transition system (such as beam search) and a scoring function for actions is known as a **policy**.) To achieve this, we can make an **early update** as soon as the oracle action sequence “falls off” the beam, even before a complete analysis is available (Collins and Roark, 2004; Daumé III and Marcu, 2005). The loss can be based on the best-scoring hypothesis on the beam, or the sum of all hypotheses (Huang et al., 2012).

For example, consider the beam search in Figure 11.7. In the correct parse, *fish* is the head of dependency arcs to both of the other two words. In the arc-standard system, this can be achieved only by using SHIFT for the first two actions. At $t = 3$, the oracle action sequence has fallen off the beam. The parser should therefore stop, and update the parameters by the gradient $\frac{\partial}{\partial \boldsymbol{\theta}} L(A_{1:3}^{(i)}, \{A_{1:3}^{(k)}\}; \boldsymbol{\theta})$, where $A_{1:3}^{(i)}$ is the first three actions of the oracle sequence, and $\{A_{1:3}^{(k)}\}$ is the beam.

This integration of incremental search and learning was first developed in the **incremental perceptron** (Collins and Roark, 2004). This method updates the parameters with respect to a hinge loss, which compares the top-scoring hypothesis and the gold action sequence, up to the current point t . Several improvements to this basic protocol are possible:

- As noted earlier, the gold dependency parse can be derived by multiple action sequences. Rather than checking for the presence of a single oracle action sequence on the beam, we can check if the gold dependency parse is *reachable* from the current beam, using a **dynamic oracle** (Goldberg and Nivre, 2012).

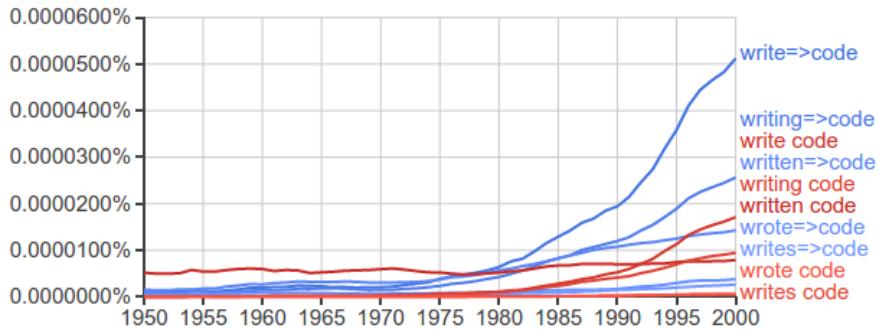


Figure 11.8: Google n-grams results for the bigram *write code* and the dependency arc *write => code* (and their morphological variants)

- By maximizing the score of the gold action sequence, we are training a decision function to find the correct action given the gold context. But in reality, the parser will make errors, and the parser is not trained to find the best action given a context that may not itself be optimal. This issue is addressed by various generalizations of incremental perceptron, known as **learning to search** (Daumé III et al., 2009). Some of these methods are discussed in chapter 15.

11.4 Applications

Dependency parsing is used in many real-world applications: any time you want to know about pairs of words which might not be adjacent, you can use dependency arcs instead of regular expression search patterns. For example, you may want to match strings like *delicious pastries*, *delicious French pastries*, and *the pastries are delicious*.

It is possible to search the Google *n*-gramscorpus by dependency edges, finding the trend in how often a dependency edge appears over time. For example, we might be interested in knowing when people started talking about *writing code*, but we also want *write some code*, *write good code*, *write all the code*, etc. The result of a search on the dependency edge *write → code* is shown in Figure 11.8. This capability has been applied to research in digital humanities, such as the analysis of gender in Shakespeare Muralidharan and Hearst (2013).

A classic application of dependency parsing is **relation extraction**, which is described

in chapter 17. The goal of relation extraction is to identify entity pairs, such as

(MELVILLE, MOBY-DICK)
 (TOLSTOY, WAR AND PEACE)
 (MARQUÉZ, 100 YEARS OF SOLITUDE)
 (SHAKESPEARE, A MIDSUMMER NIGHT'S DREAM),

6044 which stand in some relation to each other (in this case, the relation is authorship). Such
 6045 entity pairs are often referenced via consistent chains of dependency relations. Therefore,
 6046 dependency paths are often a useful feature in supervised systems which learn to detect
 6047 new instances of a relation, based on labeled examples of other instances of the same
 6048 relation type (Culotta and Sorensen, 2004; Fundel et al., 2007; Mintz et al., 2009).

6049 Cui et al. (2005) show how dependency parsing can improve automated question an-
 6050 swering. Suppose you receive the following query:

6051 (11.1) What percentage of the nation's cheese does Wisconsin produce?

6052 The corpus contains this sentence:

6053 (11.2) In Wisconsin, where farmers produce 28% of the nation's cheese, ...

6054 The location of *Wisconsin* in the surface form of this string makes it a poor match for the
 6055 query. However, in the dependency graph, there is an edge from *produce* to *Wisconsin* in
 6056 both the question and the potential answer, raising the likelihood that this span of text is
 6057 relevant to the question.

6058 A final example comes from sentiment analysis. As discussed in chapter 4, the polarity
 6059 of a sentence can be reversed by negation, e.g.

6060 (11.3) *There is no reason at all to believe the polluters will suddenly become reasonable.*

6061 By tracking the sentiment polarity through the dependency parse, we can better iden-
 6062 tify the overall polarity of the sentence, determining when key sentiment words are re-
 6063 versed (Wilson et al., 2005; Nakagawa et al., 2010).

6064 Additional resources

6065 More details on dependency grammar and parsing algorithms can be found in the manuscript
 6066 by Kübler et al. (2009). For a comprehensive but whimsical overview of graph-based de-
 6067 pendency parsing algorithms, see Eisner (1997). Jurafsky and Martin (2018) describe an
 6068 **agenda-based** version of beam search, in which the beam contains hypotheses of varying
 6069 lengths. New hypotheses are added to the beam only if their score is better than the worst

item currently on the beam. Another search algorithm for transition-based parsing is **easy-first**, which abandons the left-to-right traversal order, and adds the highest-scoring edges first, regardless of where they appear (Goldberg and Elhadad, 2010). Goldberg et al. (2013) note that although transition-based methods can be implemented in linear time in the length of the input, naïve implementations of beam search will require quadratic time, due to the cost of copying each hypothesis when it is expanded on the beam. This issue can be addressed by using a more efficient data structure for the stack.

6077 Exercises

- 6078 1. The dependency structure $1 \leftarrow 2 \rightarrow 3$, with 2 as the root, can be obtained from more
6079 than one set of actions in arc-standard parsing. List both sets of actions that can
6080 obtain this parse.
- 6081 2. Suppose you have a set of unlabeled arc scores $\psi(i \rightarrow j)$, where the score depends
6082 only on the identity of the two words. The scores include $\psi(\text{ROOT} \rightarrow j)$.
 - 6083 • Assuming each word occurs only once in the sentence ($(i \neq j) \Leftarrow (w_i \neq w_j)$),
6084 how would you construct a weighted lexicalized context-free grammar so that
6085 the score of *any* projective dependency tree is equal to the score of some equiv-
6086 alent derivation in the lexicalized context-free grammar?
 - 6087 • Verify that your method works for a simple example like *they eat fish*.
 - 6088 • How would you adapt your method to handle the case an individual word
6089 may appear multiple times in the sentence?
- 6090 3. Provide the UD-style dependency parse for the sentence *Xi-Lan eats shoots and leaves*,
6091 assuming *leaves* is a verb. Provide arc-standard and arc-eager derivations for this
6092 dependency parse.

6093

Part III

6094

Meaning

6095 Chapter 12

6096 Logical semantics

6097 The previous few chapters have focused on building systems that reconstruct the **syntax**
6098 of natural language — its structural organization — through tagging and parsing. But
6099 some of the most exciting and promising potential applications of language technology
6100 involve going beyond syntax to **semantics** — the underlying meaning of the text:

- 6101 • Answering questions, such as *where is the nearest coffeeshop?* or *what is the middle name*
6102 *of the mother of the 44th President of the United States?*.
- 6103 • Building a robot that can follow natural language instructions to execute tasks.
- 6104 • Translating a sentence from one language into another, while preserving the under-
6105 lying meaning.
- 6106 • Fact-checking an article by searching the web for contradictory evidence.
- 6107 • Logic-checking an argument by identifying contradictions, ambiguity, and unsup-
6108 ported assertions.

6109 Semantic analysis involves converting natural language into a **meaning representa-**
6110 **tion**. To be useful, a meaning representation must meet several criteria:

- 6111 • **c1**: it should be unambiguous: unlike natural language, there should be exactly one
6112 meaning per statement;
- 6113 • **c2**: it should provide a way to link language to external knowledge, observations,
6114 and actions;
- 6115 • **c3**: it should support computational **inference**, so that meanings can be combined
6116 to derive additional knowledge;
- 6117 • **c4**: it should be expressive enough to cover the full range of things that people talk
6118 about in natural language.

6119 Much more than this can be said about the question of how best to represent knowledge
 6120 for computation (e.g., Sowa, 2000), but this chapter will focus on these four criteria.

6121 12.1 Meaning and denotation

6122 The first criterion for a meaning representation is that statements in the representation
 6123 should be unambiguous — they should have only one possible interpretation. Natural
 6124 language does not have this property: as we saw in chapter 10, sentences like *cats scratch*
 6125 *people with claws* have multiple interpretations.

6126 But what does it mean for a statement to be unambiguous? Programming languages
 6127 provide a useful example: the output of a program is completely specified by the rules of
 6128 the language and the properties of the environment in which the program is run. For ex-
 6129 ample, the python code $5 + 3$ will have the output 8, as will the codes $(4 * 4) - (3 * 3) + 1$
 6130 and $((8))$. This output is known as the **denotation** of the program, and can be written
 6131 as,

$$\llbracket 5+3 \rrbracket = \llbracket (4 * 4) - (3 * 3) + 1 \rrbracket = \llbracket ((8)) \rrbracket = 8. \quad [12.1]$$

6132 The denotations of these arithmetic expressions are determined by the meaning of the
 6133 **constants** (e.g., 5, 3) and the **relations** (e.g., $+$, $*$, $(,)$). Now let's consider another snippet
 6134 of python code, `double(4)`. The denotation of this code could be, $\llbracket \text{double}(4) \rrbracket = 8$, or
 6135 it could be $\llbracket \text{double}(4) \rrbracket = 44$ — it depends on the meaning of `double`. This meaning
 6136 is defined in a **world model** \mathcal{M} as an infinite set of pairs. We write the denotation with
 6137 respect to model \mathcal{M} as $\llbracket \cdot \rrbracket_{\mathcal{M}}$, e.g., $\llbracket \text{double} \rrbracket_{\mathcal{M}} = \{(0, 0), (1, 2), (2, 4), \dots\}$. The world
 6138 model would also define the (infinite) list of constants, e.g., $\{0, 1, 2, \dots\}$. As long as the
 6139 denotation of string ϕ in model \mathcal{M} can be computed unambiguously, the language can be
 6140 said to be unambiguous.

6141 This approach to meaning is known as **model-theoretic semantics**, and it addresses
 6142 not only criterion *c1* (no ambiguity), but also *c2* (connecting language to external knowl-
 6143 edge, observations, and actions). For example, we can connect a representation of the
 6144 meaning of a statement like *the capital of Georgia* with a world model that includes knowl-
 6145 edge base of geographical facts, obtaining the denotation *Atlanta*. We might populate
 6146 a world model by applying an image analysis algorithm to Figure 12.1, and then use this
 6147 world model to evaluate **propositions** like *a man is riding a moose*. Another desirable prop-
 6148 erty of model-theoretic semantics is that when the facts change, the denotations change
 6149 too: the meaning representation of *President of the USA* would have a different denotation
 6150 in the model \mathcal{M}_{2014} as it would in \mathcal{M}_{2022} .



Figure 12.1: A (doctored) image, which could be the basis for a world model

6151 12.2 Logical representations of meaning

6152 Criterion *c3* requires that the meaning representation support inference — for example,
 6153 automatically deducing new facts from known premises. While many representations
 6154 have been proposed that meet these criteria, the most mature is the language of first-order
 6155 logic.¹

6156 12.2.1 Propositional logic

6157 The bare bones of logical meaning representation are Boolean operations on propositions:

6158 **Propositional symbols.** Greek symbols like ϕ and ψ will be used to represent **proposi-**
 6159 **tions**, which are statements that are either true or false. For example, ϕ may corre-
 6160 spond to the proposition, *bagels are delicious*.

6161 **Boolean operators.** We can build up more complex propositional formulas from Boolean
 6162 operators. These include:

- 6163 • Negation $\neg\phi$, which is true if ϕ is false.

¹Alternatives include the “variable-free” representation used in semantic parsing of geographical queries (Zelle and Mooney, 1996) and robotic control (Ge and Mooney, 2005), and dependency-based compositional semantics (Liang et al., 2013).

- 6164 • Conjunction, $\phi \wedge \psi$, which is true if both ϕ and ψ are true.
- 6165 • Disjunction, $\phi \vee \psi$, which is true if at least one of ϕ and ψ is true
- 6166 • Implication, $\phi \Rightarrow \psi$, which is true unless ϕ is true and ψ is false. Implication
- 6167 has identical truth conditions to $\neg\phi \vee \psi$.
- 6168 • Equivalence, $\phi \Leftrightarrow \psi$, which is true if ϕ and ψ are both true or both false. Equiv-
- 6169 alence has identical truth conditions to $(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$.

6170 It is not strictly necessary to have all five Boolean operators: readers familiar with
 6171 Boolean logic will know that it is possible to construct all other operators from either the
 6172 NAND (not-and) or NOR (not-or) operators. Nonetheless, it is clearest to use all five
 6173 operators. From the truth conditions for these operators, it is possible to define a number
 6174 of “laws” for these Boolean operators, such as,

- 6175 • *Commutativity*: $\phi \wedge \psi = \psi \wedge \phi$, $\phi \vee \psi = \psi \vee \phi$
- 6176 • *Associativity*: $\phi \wedge (\psi \wedge \chi) = (\phi \wedge \psi) \wedge \chi$, $\phi \vee (\psi \vee \chi) = (\phi \vee \psi) \vee \chi$
- 6177 • *Complementation*: $\phi \wedge \neg\phi = \perp$, $\phi \vee \neg\phi = \top$, where \top indicates a true proposition
 6178 and \perp indicates a false proposition.

These laws can be combined to derive further equivalences, which can support logical inferences. For example, suppose $\phi = \text{The music is loud}$ and $\psi = \text{Max can't sleep}$. Then if we are given,

$$\begin{aligned} \phi \Rightarrow \psi & \quad \text{If the music is loud, Max can't sleep.} \\ \phi & \quad \text{The music is loud.} \end{aligned}$$

6179 we can derive ψ (*Max can't sleep*) by application of **modus ponens**, which is one of a
 6180 set of **inference rules** that can be derived from more basic laws and used to manipulate
 6181 propositional formulas. **Automated theorem provers** are capable of applying inference
 6182 rules to a set of premises to derive desired propositions (Loveland, 2016).

6183 12.2.2 First-order logic

6184 Propositional logic is so named because it treats propositions as its base units. However,
 6185 the criterion *c4* states that our meaning representation should be sufficiently expressive.
 6186 Now consider the sentence pair,

- 6187 (12.1) If anyone is making noise, then Max can't sleep.
 6188 Abigail is making noise.

6189 People are capable of making inferences from this sentence pair, but such inferences re-
 6190 quire formal tools that are beyond propositional logic. To understand the relationship

6191 between the statement *anyone is making noise* and the statement *Abigail is making noise*, our
 6192 meaning representation requires the additional machinery of **first-order logic** (FOL).

6193 In FOL, logical propositions can be constructed from relationships between entities.
 6194 Specifically, FOL extends propositional logic with the following classes of terms:

6195 **Constants.** These are elements that name individual entities in the model, such as MAX
 6196 and ABIGAIL. The denotation of each constant in a model \mathcal{M} is an element in the
 6197 model, e.g., $[\![\text{MAX}]\!] = m$ and $[\![\text{ABIGAIL}]\!] = a$.

6198 **Relations.** Relations can be thought of as sets of entities, or sets of tuples. For example,
 6199 the relation CAN-SLEEP is defined as the set of entities who can sleep, and has the
 6200 denotation $[\![\text{CAN-SLEEP}]\!] = \{a, m, \dots\}$. To test the truth value of the proposition
 6201 CAN-SLEEP(MAX), we ask whether $[\![\text{MAX}]\!] \in [\![\text{CAN-SLEEP}]\!]$. Logical relations that are
 6202 defined over sets of entities are sometimes called **properties**.

6203 Relations may also be ordered tuples of entities. For example BROTHER(MAX,ABIGAIL)
 6204 expresses the proposition that MAX is the brother of ABIGAIL. The denotation of
 6205 such relations is a set of tuples, $[\![\text{BROTHER}]\!] = \{(m, a), (x, y), \dots\}$. To test the
 6206 truth value of the proposition BROTHER(MAX,ABIGAIL), we ask whether the tuple
 6207 $([\![\text{MAX}]\!], [\![\text{ABIGAIL}]\!])$ is in the denotation $[\![\text{BROTHER}]\!]$.

Using constants and relations, it is possible to express statements like *Max can't sleep* and *Max is Abigail's brother*:

$$\neg \text{CAN-SLEEP}(\text{MAX}) \\ \text{BROTHER}(\text{MAX}, \text{ABIGAIL}).$$

These statements can also be combined using Boolean operators, such as,

$$(\text{BROTHER}(\text{MAX}, \text{ABIGAIL}) \vee \text{BROTHER}(\text{MAX}, \text{STEVE})) \Rightarrow \neg \text{CAN-SLEEP}(\text{MAX}).$$

6208 This fragment of first-order logic permits only statements about specific entities. To
 6209 support inferences about statements like *If anyone is making noise, then Max can't sleep*,
 6210 two more elements must be added to the meaning representation:

6211 **Variables.** Variables are mechanisms for referring to entities that are not locally specified.
 6212 We can then write $\text{CAN-SLEEP}(x)$ or $\text{BROTHER}(x, \text{ABIGAIL})$. In these cases, x is a **free
 6213 variable**, meaning that we have not committed to any particular assignment.

6214 **Quantifiers.** Variables are bound by quantifiers. There are two quantifiers in first-order
 6215 logic.²

- 6216 • The **existential quantifier** \exists , which indicates that there must be at least one en-
 6217 tity to which the variable can bind. For example, the statement $\exists x \text{MAKES-NOISE}(x)$
 6218 indicates that there is at least one entity for which MAKES-NOISE is true.
 6219 • The **universal quantifier** \forall , which indicates that the variable must be able to
 6220 bind to any entity in the model. For example, the statement,

$$\text{MAKES-NOISE(ABIGAIL)} \Rightarrow (\forall x \neg \text{CAN-SLEEP}(x)) \quad [12.3]$$

6221 asserts that if Abigail makes noise, no one can sleep.

6222 The expressions $\exists x$ and $\forall x$ make x into a **bound variable**. A formula that contains
 6223 no free variables is a **sentence**.

6224 **Functions.** Functions map from entities to entities, e.g., $\llbracket \text{CAPITAL-OF(GEORGIA)} \rrbracket = \llbracket \text{ATLANTA} \rrbracket$.
 6225 With functions, it is convenient to add an equality operator, supporting statements
 6226 like,

$$\forall x \exists y \text{MOTHER-OF}(x) = \text{DAUGHTER-OF}(y). \quad [12.4]$$

6227 Note that MOTHER-OF is a functional analogue of the relation MOTHER, so that
 6228 $\text{MOTHER-OF}(x) = y$ if $\text{MOTHER}(x, y)$. Any logical formula that uses functions can be
 6229 rewritten using only relations and quantification. For example,

$$\text{MAKES-NOISE}(\text{MOTHER-OF(ABIGAIL)}) \quad [12.5]$$

6230 can be rewritten as $\exists x \text{MAKES-NOISE}(x) \wedge \text{MOTHER}(x, \text{ABIGAIL})$.

An important property of quantifiers is that the order can matter. Unfortunately, natural language is rarely clear about this! The issue is demonstrated by examples like *everyone speaks a language*, which has the following interpretations:

$$\forall x \exists y \text{ SPEAKS}(x, y) \quad [12.6]$$

$$\exists y \forall x \text{ SPEAKS}(x, y). \quad [12.7]$$

6231 In the first case, y may refer to several different languages, while in the second case, there
 6232 is a single y that is spoken by everyone.

²In first-order logic, it is possible to quantify only over entities. In **second-order logic**, it is possible to quantify over properties, supporting statements like *Butch has every property that a good boxer has* (example from Blackburn and Bos, 2005),

$$\forall P \forall x ((\text{GOOD-BOXER}(x) \Rightarrow P(x)) \Rightarrow P(\text{BUTCH})). \quad [12.2]$$

6233 12.2.2.1 Truth-conditional semantics

6234 One way to look at the meaning of an FOL sentence ϕ is as a set of **truth conditions**,
 6235 or models under which ϕ is satisfied. But how to determine whether a sentence is true
 6236 or false in a given model? We will approach this inductively, starting with a predicate
 6237 applied to a tuple of constants. The truth of such a sentence depends on whether the
 6238 tuple of denotations of the constants is in the denotation of the predicate. For example,
 6239 CAPITAL(GEORGIA,ATLANTA) is true in model \mathcal{M} iff,

$$(\llbracket \text{GEORGIA} \rrbracket_{\mathcal{M}}, \llbracket \text{ATLANTA} \rrbracket_{\mathcal{M}}) \in \llbracket \text{CAPITAL} \rrbracket_{\mathcal{M}}. \quad [12.8]$$

6240 The Boolean operators \wedge, \vee, \dots provide ways to construct more complicated sentences,
 6241 and the truth of such statements can be assessed based on the truth tables associated with
 6242 these operators. The statement $\exists x\phi$ is true if there is some assignment of the variable x
 6243 to an entity in the model such that ϕ is true; the statement $\forall x\phi$ is true if ϕ is true under
 6244 all possible assignments of x . More formally, we would say that ϕ is **satisfied** under \mathcal{M} ,
 6245 written as $\mathcal{M} \models \phi$.

6246 Truth conditional semantics allows us to define several other properties of sentences
 6247 and pairs of sentences. Suppose that in every \mathcal{M} under which ϕ is satisfied, another
 6248 formula ψ is also satisfied; then ϕ **entails** ψ , which is also written as $\phi \models \psi$. For example,

$$\text{CAPITAL(GEORGIA,ATLANTA)} \models \exists x \text{CAPITAL(GEORGIA, } x\text{)}. \quad [12.9]$$

6249 A statement that is satisfied under any model, such as $\phi \vee \neg\phi$, is **valid**, written $\models (\phi \vee$
 6250 $\neg\phi)$. A statement that is not satisfied under any model, such as $\phi \wedge \neg\phi$, is **unsatisfiable**,
 6251 or **inconsistent**. A **model checker** is a program that determines whether a sentence ϕ
 6252 is satisfied in \mathcal{M} . A **model builder** is a program that constructs a model in which ϕ
 6253 is satisfied. The problems of checking for consistency and validity in first-order logic
 6254 are **undecidable**, meaning that there is no algorithm that can automatically determine
 6255 whether an FOL formula is valid or inconsistent.

6256 12.2.2.2 Inference in first-order logic

6257 Our original goal was to support inferences that combine general statements *If anyone is*
making noise, then Max can't sleep with specific statements like *Abigail is making noise*. We
 6258 can now represent such statements in first-order logic, but how are we to perform the
 6259 inference that *Max can't sleep*? One approach is to use “generalized” versions of proposi-
 6260 tional inference rules like modus ponens, which can be applied to FOL formulas. By
 6261 repeatedly applying such inference rules to a knowledge base of facts, it is possible to
 6262 produce proofs of desired propositions. To find the right sequence of inferences to derive
 6263 a desired theorem, classical artificial intelligence search algorithms like backward chain-
 6264 ing can be applied. Such algorithms are implemented in interpreters for the `prolog` logic
 6265 programming language (Pereira and Shieber, 2002).

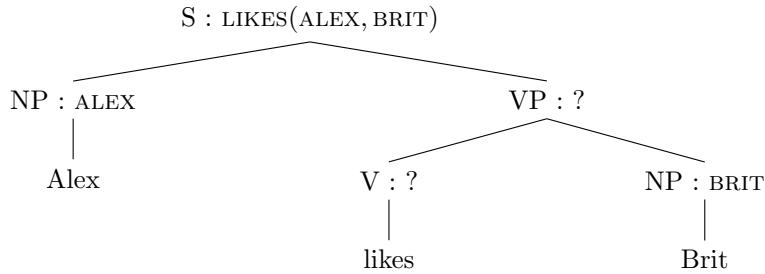


Figure 12.2: The principle of compositionality requires that we identify meanings for the constituents *likes* and *likes Brit* that will make it possible to compute the meaning for the entire sentence.

6267 12.3 Semantic parsing and the lambda calculus

6268 The previous section laid out a lot of formal machinery; the remainder of this chapter
 6269 links these formalisms back to natural language. Given an English sentence like *Alex likes*
 6270 *Brit*, how can we obtain the desired first-order logical representation, $\text{LIKES}(\text{ALEX}, \text{BRIT})$?
 6271 This is the task of **semantic parsing**. Just as a syntactic parser is a function from a natu-
 6272 ral language sentence to a syntactic structure such as a phrase structure tree, a semantic
 6273 parser is a function from natural language to logical formulas.

6274 As in syntactic analysis, semantic parsing is difficult because the space of inputs and
 6275 outputs is very large, and their interaction is complex. Our best hope is that, like syntactic
 6276 parsing, semantic parsing can somehow be decomposed into simpler sub-problems. This
 6277 idea, usually attributed to the German philosopher Gottlob Frege, is called the **principle**
 6278 **of compositionality**: the meaning of a complex expression is a function of the meanings of
 6279 that expression's constituent parts. We will define these “constituent parts” as syntactic
 6280 constituents: noun phrases and verb phrases. These constituents are combined using
 6281 function application: if the syntactic parse contains the production $x \rightarrow y z$, then the
 6282 semantics of x , written $x.\text{sem}$, will be computed as a function of the semantics of the
 6283 constituents, $y.\text{sem}$ and $z.\text{sem}$.³ ⁴

³§ 9.3.2 briefly discusses Combinatory Categorial Grammar (CCG) as an alternative to a phrase-structure analysis of syntax. CCG is argued to be particularly well-suited to semantic parsing (Hockenmaier and Steedman, 2007), and is used in much of the contemporary work on machine learning for semantic parsing, summarized in § 12.4.

⁴The approach of algorithmically building up meaning representations from a series of operations on the syntactic structure of a sentence is generally attributed to the philosopher Richard Montague, who published a series of influential papers on the topic in the early 1970s (e.g., Montague, 1973).

6284 **12.3.1 The lambda calculus**

6285 Let's see how this works for a simple sentence like *Alex likes Brit*, whose syntactic structure
 6286 is shown in Figure 12.2. Our goal is the formula, LIKES(ALEX,BRIT), and it is clear that the
 6287 meaning of the constituents *Alex* and *Brit* should be ALEX and BRIT. That leaves two more
 6288 constituents: the verb *likes*, and the verb phrase *likes Brit*. The meanings of these units
 6289 must be defined in a way that makes it possible to recover the desired meaning for the
 6290 entire sentence by function application. If the meanings of *Alex* and *Brit* are constants,
 6291 then the meanings of *likes* and *likes Brit* must be functional expressions, which can be
 6292 applied to their siblings to produce the desired analyses.

6293 Modeling these partial analyses requires extending the first-order logic meaning rep-
 6294 resentation. We do this by adding **lambda expressions**, which are descriptions of anonym-
 6295 ous functions,⁵ e.g.,

$$\lambda x.\text{LIKES}(x, \text{BRIT}). \quad [12.10]$$

6296 This functional expression is the meaning of the verb phrase *likes Brit*; it takes a single
 6297 argument, and returns the result of substituting that argument for x in the expression
 6298 $\text{LIKES}(x, \text{BRIT})$. We write this substitution as,

$$(\lambda x.\text{LIKES}(x, \text{BRIT}))@\text{ALEX} = \text{LIKES}(\text{ALEX}, \text{BRIT}), \quad [12.11]$$

6299 with the symbol "@" indicating function application. Function application in the lambda
 6300 calculus is sometimes called **β -reduction** or **β -conversion**. The expression $\phi@\psi$ indicates
 6301 a function application to be performed by β -reduction, and $\phi(\psi)$ indicates a function or
 6302 predicate in the final logical form.

6303 Equation 12.11 shows how to obtain the desired semantics for the sentence *Alex likes*
 6304 *Brit*: by applying the lambda expression $\lambda x.\text{LIKES}(x, \text{BRIT})$ to the logical constant ALEX.
 6305 This rule of composition can be specified in a **syntactic-semantic grammar**, in which
 6306 syntactic productions are paired with semantic operations. For the syntactic production
 6307 $S \rightarrow NP VP$, we have the semantic rule $VP.sem@NP.sem$.

The meaning of the transitive verb phrase *likes Brit* can also be obtained by function
 application on its syntactic constituents. For the syntactic production $VP \rightarrow V NP$, we
 apply the semantic rule,

$$VP.sem = (V.sem)@NP.sem \quad [12.12]$$

$$= (\lambda y.\lambda x.\text{LIKES}(x, y))@(\text{BRIT}) \quad [12.13]$$

$$= \lambda x.\text{LIKES}(x, \text{BRIT}). \quad [12.14]$$

⁵Formally, all first-order logic formulas are lambda expressions; in addition, if ϕ is a lambda expression, then $\lambda x.\phi$ is also a lambda expression. Readers who are familiar with functional programming will recognize lambda expressions from their use in programming languages such as Lisp and Python.

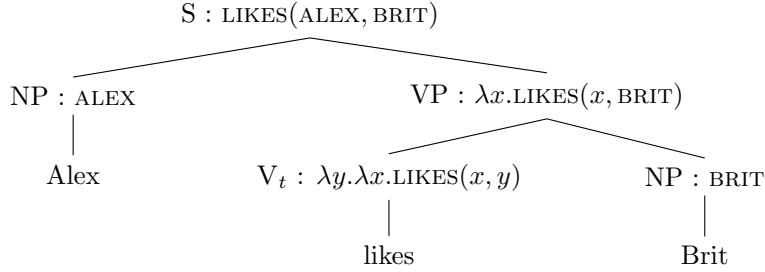


Figure 12.3: Derivation of the semantic representation for *Alex likes Brit* in the grammar G_1 .

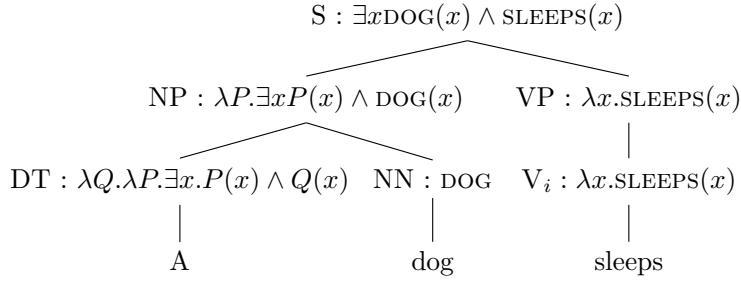
S	\rightarrow	NP VP	VP.sem@NP.sem
VP	\rightarrow	V _t NP	V _t .sem@NP.sem
VP	\rightarrow	V _i	V _i .sem
V _t	\rightarrow	likes	$\lambda y. \lambda x. \text{LIKES}(x, y)$
V _i	\rightarrow	sleeps	$\lambda x. \text{SLEEPS}(x)$
NP	\rightarrow	Alex	ALEX
NP	\rightarrow	Brit	BRIT

Table 12.1: G_1 , a minimal syntactic-semantic context-free grammar

6308 Thus, the meaning of the transitive verb *likes* is a lambda expression whose output is
 6309 *another* lambda expression: it takes y as an argument to fill in one of the slots in the LIKES
 6310 relation, and returns a lambda expression that is ready to take an argument to fill in the
 6311 other slot.⁶

6312 Table 12.1 shows a minimal syntactic-semantic grammar fragment, G_1 . The complete
 6313 **derivation** of *Alex likes Brit* in G_1 is shown in Figure 12.3. In addition to the transitive
 6314 verb *likes*, the grammar also includes the intransitive verb *sleeps*; it should be clear how
 6315 to derive the meaning of sentences like *Alex sleeps*. For verbs that can be either transitive
 6316 or intransitive, such as *eats*, we would have two terminal productions, one for each sense
 6317 (terminal productions are also called the **lexical entries**). Indeed, most of the grammar is
 6318 in the **lexicon** (the terminal productions), since these productions select the basic units of
 6319 the semantic interpretation.

⁶This can be written in a few different ways. The notation $\lambda y. x. \text{LIKES}(x, y)$ is a somewhat informal way to indicate a lambda expression that takes two arguments; this would be acceptable in functional programming. Logicians (e.g., Carpenter, 1997) often prefer the more formal notation $\lambda y. \lambda x. \text{LIKES}(x)(y)$, indicating that each lambda expression takes exactly one argument.

Figure 12.4: Derivation of the semantic representation for *A dog sleeps*, in grammar G_2

6320 **12.3.2 Quantification**

6321 Things get more complicated when we move from sentences about named entities to sen-
 6322 tences that involve more general noun phrases. Let's consider the example, *A dog sleeps*,
 6323 which has the meaning $\exists x \text{DOG}(x) \wedge \text{SLEEPS}(x)$. Clearly, the DOG relation will be intro-
 6324 duced by the word *dog*, and the SLEEP relation will be introduced by the word *sleeps*.⁷
 6325 The existential quantifier \exists must be introduced by the lexical entry for the determiner *a*.⁷
 6326 However, this seems problematic for the compositional approach taken in the grammar
 6327 G_1 : if the semantics of the noun phrase *a dog* is an existentially quantified expression, how
 6328 can it be the argument to the semantics of the verb *sleeps*, which expects an entity? And
 6329 where does the logical conjunction come from?

6330 There are a few different approaches to handling these issues.⁸ We will begin by re-
 6331 versing the semantic relationship between subject NPs and VPs, so that the production
 6332 $S \rightarrow \text{NP VP}$ has the semantics $\text{NP.sem} @ \text{VP.sem}$: the meaning of the sentence is now the
 6333 semantics of the noun phrase applied to the verb phrase. The implications of this change
 6334 are best illustrated by exploring the derivation of the example, shown in Figure 12.4. Let's
 6335 start with the indefinite article *a*, to which we assign the rather intimidating semantics,

$$\lambda P. \lambda Q. \exists x P(x) \wedge Q(x). \quad [12.15]$$

This is a lambda expression that takes two **relations** as arguments, P and Q . The relation P is scoped to the outer lambda expression, so it will be provided by the immediately

⁷Conversely, the sentence *Every dog sleeps* would involve a universal quantifier, $\forall x \text{DOG}(x) \Rightarrow \text{SLEEPS}(x)$. The definite article *the* requires more consideration, since *the dog* must refer to some dog which is uniquely identifiable, perhaps from contextual information external to the sentence. Carpenter (1997, pp. 96-100) summarizes recent approaches to handling definite descriptions.

⁸Carpenter (1997) offers an alternative treatment based on combinatory categorial grammar.

adjacent noun, which in this case is DOG. Thus, the noun phrase *a dog* has the semantics,

$$\text{NP.sem} = \text{DET.sem} @ \text{NN.sem} \quad [12.16]$$

$$= (\lambda P. \lambda Q. \exists x P(x) \wedge Q(x)) @ (\text{DOG}) \quad [12.17]$$

$$= \lambda Q. \exists x \text{DOG}(x) \wedge Q(x). \quad [12.18]$$

6336 This is a lambda expression that is expecting another relation, Q , which will be provided
 6337 by the verb phrase, SLEEPS. This gives the desired analysis, $\exists x \text{DOG}(x) \wedge \text{SLEEPS}(x)$.⁹

6338 If noun phrases like *a dog* are interpreted as lambda expressions, then proper nouns
 6339 like *Alex* must be treated in the same way. This is achieved by **type-raising** from con-
 6340 stants to lambda expressions, $x \Rightarrow \lambda P. P(x)$. After type-raising, the semantics of *Alex* is
 6341 $\lambda P. P(\text{ALEX})$ — a lambda expression that expects a relation to tell us something about
 6342 *ALEX*.¹⁰ Again, make sure you see how the analysis in Figure 12.4 can be applied to the
 6343 sentence *Alex sleeps*.

6344 Direct objects are handled by applying the same type-raising operation to transitive
 6345 verbs: the meaning of verbs such as *likes* is raised to,

$$\lambda P. \lambda x. P(\lambda y. \text{LIKES}(x, y)) \quad [12.19]$$

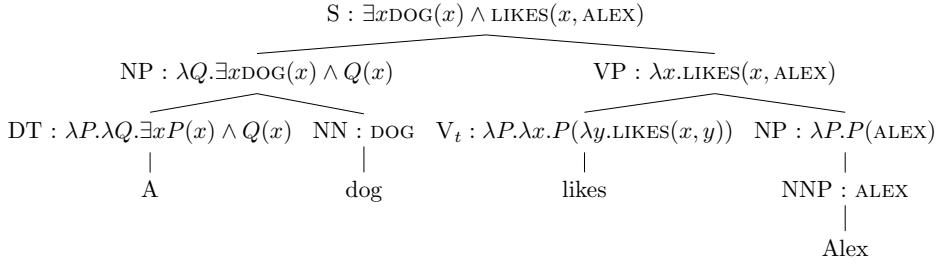
As a result, we can keep the verb phrase production $\text{VP.sem} = \text{V.sem} @ \text{NP.sem}$, knowing
 that the direct object will provide the function P in Equation 12.19. To see how this works,
 let's analyze the verb phrase *likes a dog*. After uniquely relabeling each lambda variable,
 we have,

$$\begin{aligned} \text{VP.sem} &= \text{V.sem} @ \text{NP.sem} \\ &= (\lambda P. \lambda x. P(\lambda y. \text{LIKES}(x, y))) @ (\lambda Q. \exists z \text{DOG}(z) \wedge Q(z)) \\ &= \lambda x. (\lambda Q. \exists z \text{DOG}(z) \wedge Q(z)) @ (\lambda y. \text{LIKES}(x, y)) \\ &= \lambda x. \exists z \text{DOG}(z) \wedge (\lambda y. \text{LIKES}(x, y)) @ z \\ &= \lambda x. \exists z \text{DOG}(z) \wedge \text{LIKES}(x, z). \end{aligned}$$

6346 These changes are summarized in the revised grammar G_2 , shown in Table 12.2. Fig-
 6347 ure 12.5 shows a derivation that involves a transitive verb, an indefinite noun phrase, and
 6348 a proper noun.

⁹When applying β -reduction to arguments that are themselves lambda expressions, be sure to use unique variable names to avoid confusion. For example, it is important to distinguish the x in the semantics for *a* from the x in the semantics for *likes*. Variable names are abstractions, and can always be changed — this is known as **α -conversion**. For example, $\lambda x. P(x)$ can be converted to $\lambda y. P(y)$, etc.

¹⁰Compositional semantic analysis is often supported by **type systems**, which make it possible to check whether a given function application is valid. The base types are entities e and truth values t . A property, such as DOG, is a function from entities to truth values, so its type is written $\langle e, t \rangle$. A transitive verb has type

Figure 12.5: Derivation of the semantic representation for *A dog likes Alex*.

S	\rightarrow NP VP	NP.sem@VP.sem
VP	\rightarrow V _t NP	V _t .sem@NP.sem
VP	\rightarrow V _i	V _i .sem
NP	\rightarrow DET NN	DET.sem@NN.sem
NP	\rightarrow NNP	$\lambda P. P(\text{NNP.sem})$
DET	$\rightarrow a$	$\lambda P. \lambda Q. \exists x P(x) \wedge Q(x)$
DET	\rightarrow every	$\lambda P. \lambda Q. \forall x (P(x) \Rightarrow Q(x))$
V _t	\rightarrow likes	$\lambda P. \lambda x. P(\lambda y. \text{LIKES}(x, y))$
V _i	\rightarrow sleeps	$\lambda x. \text{SLEEPS}(x)$
NN	\rightarrow dog	DOG
NNP	\rightarrow Alex	ALEX
NNP	\rightarrow Brit	BRIT

Table 12.2: G_2 , a syntactic-semantic context-free grammar fragment, which supports quantified noun phrases

6349 12.4 Learning semantic parsers

6350 As with syntactic parsing, any syntactic-semantic grammar with sufficient coverage risks
 6351 producing many possible analyses for any given sentence. Machine learning is the dom-
 6352 inant approach to selecting a single analysis. We will focus on algorithms that learn to
 6353 score logical forms by attaching weights to features of their derivations (Zettlemoyer
 6354 and Collins, 2005). Alternative approaches include transition-based parsing (Zelle and
 6355 Mooney, 1996; Misra and Artzi, 2016) and methods inspired by machine translation (Wong
 6356 and Mooney, 2006). Methods also differ in the form of supervision used for learning,

$\langle e, \langle e, t \rangle \rangle$: after receiving the first entity (the direct object), it returns a function from entities to truth values, which will be applied to the subject of the sentence. The type-raising operation $x \Rightarrow \lambda P. P(x)$ corresponds to a change in type from e to $\langle \langle e, t \rangle, t \rangle$: it expects a function from entities to truth values, and returns a truth value.

which can range from complete derivations to much more limited training signals. We will begin with the case of complete supervision, and then consider how learning is still possible even when seemingly key information is missing.

Datasets Early work on semantic parsing focused on natural language expressions of geographical database queries, such as *What states border Texas*. The GeoQuery dataset of Zelle and Mooney (1996) was originally coded in prolog, but has subsequently been expanded and converted into the SQL database query language by Popescu et al. (2003) and into first-order logic with lambda calculus by Zettlemoyer and Collins (2005), providing logical forms like $\lambda x.\text{STATE}(x) \wedge \text{BORDERS}(x, \text{TEXAS})$. Another early dataset consists of instructions for RoboCup robot soccer teams (Kate et al., 2005). More recent work has focused on broader domains, such as the Freebase database (Bollacker et al., 2008), for which queries have been annotated by Krishnamurthy and Mitchell (2012) and Cai and Yates (2013). Other recent datasets include child-directed speech (Kwiatkowski et al., 2012) and elementary school science exams (Krishnamurthy, 2016).

12.4.1 Learning from derivations

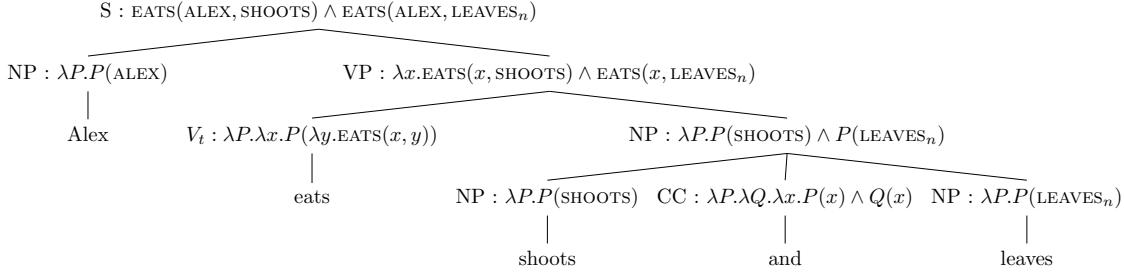
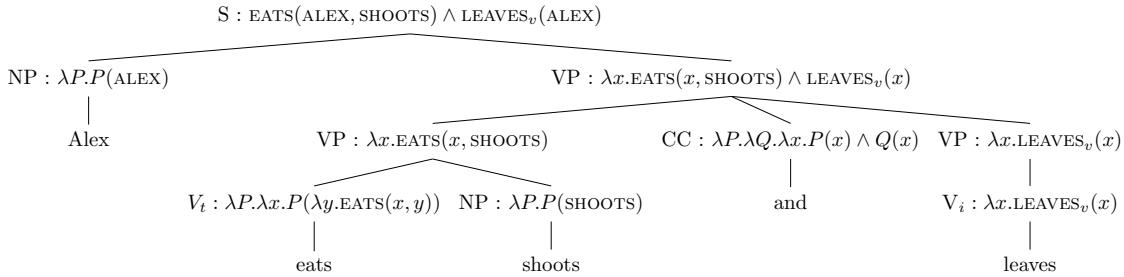
Let $w^{(i)}$ indicate a sequence of text, and let $y^{(i)}$ indicate the desired logical form. For example:

$$\begin{aligned} w^{(i)} &= \text{Alex eats shoots and leaves} \\ y^{(i)} &= \text{EATS(ALEX,SHOOTS)} \wedge \text{EATS(ALEX,LEAVES)} \end{aligned}$$

In the standard supervised learning paradigm that was introduced in § 2.2, we first define a feature function, $f(w, y)$, and then learn weights on these features, so that $y^{(i)} = \operatorname{argmax}_y \theta \cdot f(w, y)$. The weight vector θ is learned by comparing the features of the true label $f(w^{(i)}, y^{(i)})$ against either the features of the predicted label $f(w^{(i)}, \hat{y})$ (perceptron, support vector machine) or the expected feature vector $E_{y|w}[f(w^{(i)}, y)]$ (logistic regression).

While this basic framework seems similar to discriminative syntactic parsing, there is a crucial difference. In (context-free) syntactic parsing, the annotation $y^{(i)}$ contains all of the syntactic productions; indeed, the task of identifying the correct set of productions is identical to the task of identifying the syntactic structure. In semantic parsing, this is not the case: the logical form $\text{EATS(ALEX,SHOOTS)} \wedge \text{EATS(ALEX,LEAVES)}$ does not reveal the syntactic-semantic productions that were used to obtain it. Indeed, there may be **spurious ambiguity**, so that a single logical form can be reached by multiple derivations. (We previously encountered spurious ambiguity in transition-based dependency parsing, § 11.3.2.)

These ideas can be formalized by introducing an additional variable z , representing the **derivation** of the logical form y from the text w . Assume that the feature function de-

Figure 12.6: Derivation for gold semantic analysis of *Alex eats shoots and leaves*Figure 12.7: Derivation for incorrect semantic analysis of *Alex eats shoots and leaves*

6389 composes across the productions in the derivation, $f(\mathbf{w}, \mathbf{z}, \mathbf{y}) = \sum_{t=1}^T f(\mathbf{w}, z_t, \mathbf{y})$, where
 6390 z_t indicates a single syntactic-semantic production. For example, we might have a feature
 6391 for the production $S \rightarrow NP VP : NP.sem@VP.sem$, as well as for terminal productions
 6392 like $NNP \rightarrow Alex : ALEX$. Under this decomposition, it is possible to compute scores
 6393 for each semantically-annotated subtree in the analysis of \mathbf{w} , so that bottom-up parsing
 6394 algorithms like CKY (§ 10.1) can be applied to find the best-scoring semantic analysis.

6395 Figure 12.6 shows a derivation of the correct semantic analysis of the sentence *Alex*
 6396 *eats shoots and leaves*, in a simplified grammar in which the plural noun phrases *shoots*
 6397 and *leaves* are interpreted as logical constants *SHOOTS* and *LEAVES_n*. Figure 12.7 shows a
 6398 derivation of an incorrect analysis. Assuming one feature per production, the perceptron
 6399 update is shown in Table 12.3. From this update, the parser would learn to prefer the
 6400 noun interpretation of *leaves* over the verb interpretation. It would also learn to prefer
 6401 noun phrase coordination over verb phrase coordination.

6402 While the update is explained in terms of the perceptron, it would be easy to replace
 6403 the perceptron with a conditional random field. In this case, the online updates would be
 6404 based on feature expectations, which can be computed using the inside-outside algorithm
 6405 (§ 10.6).

$NP_1 \rightarrow NP_2 \ CC \ NP_3$	$(CC.sem @ (NP_2.sem)) @ (NP_3.sem)$	+1
$VP_1 \rightarrow VP_2 \ CC \ VP_3$	$(CC.sem @ (VP_2.sem)) @ (VP_3.sem)$	-1
$NP \rightarrow leaves$	$LEAVES_n$	+1
$VP \rightarrow V_i$	$V_i.sem$	-1
$V_i \rightarrow leaves$	$\lambda x.LEAVES_v$	-1

Table 12.3: Perceptron update for analysis in Figure 12.6 (gold) and Figure 12.7 (predicted)

6406 **12.4.2 Learning from logical forms**

Complete derivations are expensive to annotate, and are rarely available.¹¹ One solution is to focus on learning from logical forms directly, while treating the derivations as **latent variables** (Zettlemoyer and Collins, 2005). In a conditional probabilistic model over logical forms y and derivations z , we have,

$$p(y, z | w) = \frac{\exp(\theta \cdot f(w, z, y))}{\sum_{y', z'} \exp(\theta \cdot f(w, z', y'))}, \quad [12.20]$$

6407 which is the standard log-linear model, applied to the logical form y and the derivation
6408 z .

Since the derivation z unambiguously determines the logical form y , it may seem silly to model the joint probability over y and z . However, since z is unknown, it can be marginalized out,

$$p(y | w) = \sum_z p(y, z | w). \quad [12.21]$$

The semantic parser can then select the logical form with the maximum log marginal probability,

$$\log \sum_z p(y, z | w) = \log \sum_z \frac{\exp(\theta \cdot f(w, z, y))}{\sum_{y', z'} \exp(\theta \cdot f(w, z', y'))} \quad [12.22]$$

$$\propto \log \sum_z \exp(\theta \cdot f(w, z', y')) \quad [12.23]$$

$$\geq \max_z \theta \cdot f(w, z, y). \quad [12.24]$$

6409 It is impossible to push the log term inside the sum over z , so our usual linear scoring
6410 function does not apply. We can recover this scoring function only in approximation, by
6411 taking the max (rather than the sum) over derivations z , which provides a lower bound.

¹¹An exception is the work of Ge and Mooney (2005), who annotate the meaning of each syntactic constituents for several hundred sentences.

Learning can be performed by maximizing the log marginal likelihood,

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^N \log p(\mathbf{y}^{(i)} \mid \mathbf{w}^{(i)}; \boldsymbol{\theta}) \quad [12.25]$$

$$= \sum_{i=1}^N \log \sum_z p(\mathbf{y}^{(i)}, \mathbf{z}^{(i)} \mid \mathbf{w}^{(i)}; \boldsymbol{\theta}). \quad [12.26]$$

6412 This log-likelihood is not **convex** in $\boldsymbol{\theta}$, unlike the log-likelihood of a fully-observed conditional random field. This means that learning can give different results depending on the
 6413 initialization.
 6414

The derivative of Equation 12.26 is,

$$\frac{\partial \ell_i}{\partial \boldsymbol{\theta}} = \sum_z p(z \mid \mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) \mathbf{f}(\mathbf{w}, z, \mathbf{y}) - \sum_{z'} p(z' \mid \mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) \mathbf{f}(\mathbf{w}, z', \mathbf{y}) \quad [12.27]$$

$$= E_{z|\mathbf{y}, \mathbf{w}} \mathbf{f}(\mathbf{w}, z, \mathbf{y}) - E_{y, z|\mathbf{w}} \mathbf{f}(\mathbf{w}, z, \mathbf{y}) \quad [12.28]$$

6415 Both expectations can be computed via bottom-up algorithms like inside-outside. Alternatively, we can again maximize rather than marginalize over derivations for an approx-
 6416 imate solution. In either case, the first term of the gradient requires us to identify
 6417 derivations z that are compatible with the logical form \mathbf{y} . This can be done in a bottom-
 6418 up dynamic programming algorithm, by having each cell in the table $t[i, j, X]$ include the
 6419 set of all possible logical forms for $X \rightsquigarrow \mathbf{w}_{i+1:j}$. The resulting table may therefore be much
 6420 larger than in syntactic parsing. This can be controlled by using pruning to eliminate inter-
 6421 mediate analyses that are incompatible with the final logical form \mathbf{y} (Zettlemoyer and
 6422 Collins, 2005), or by using beam search and restricting the size of each cell to some fixed
 6423 constant (Liang et al., 2013).
 6424

6425 If we replace each expectation in Equation 12.28 with argmax and then apply stochastic
 6426 gradient descent to learn the weights, we obtain the **latent variable perceptron**, a simple
 6427 and general algorithm for learning with missing data. The algorithm is shown in its most
 6428 basic form in Algorithm 16, but the usual tricks such as averaging and margin loss can
 6429 be applied (Yu and Joachims, 2009). Aside from semantic parsing, the latent variable
 6430 perceptron has been used in tasks such as machine translation (Liang et al., 2006) and
 6431 named entity recognition (Sun et al., 2009). In **latent conditional random fields**, we use
 6432 the full expectations rather than maximizing over the hidden variable. This model has
 6433 also been employed in a range of problems beyond semantic parsing, including parse
 6434 reranking (Koo and Collins, 2005) and gesture recognition (Quattoni et al., 2007).

6435 12.4.3 Learning from denotations

Logical forms are easier to obtain than complete derivations, but the annotation of logical forms still requires considerable expertise. However, it is relatively easy to obtain deno-

Algorithm 16 Latent variable perceptron

```

1: procedure LATENTVARIABLEPERCEPTRON( $\mathbf{w}^{(1:N)}, \mathbf{y}^{(1:N)}$ )
2:    $\theta \leftarrow 0$ 
3:   repeat
4:     Select an instance  $i$ 
5:      $\mathbf{z}^{(i)} \leftarrow \text{argmax}_{\mathbf{z}} \theta \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{z}, \mathbf{y}^{(i)})$ 
6:      $\hat{\mathbf{y}}, \hat{\mathbf{z}} \leftarrow \text{argmax}_{\mathbf{y}', \mathbf{z}'} \theta \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{z}', \mathbf{y}')$ 
7:      $\theta \leftarrow \theta + \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{z}^{(i)}, \mathbf{y}^{(i)}) - \mathbf{f}(\mathbf{w}^{(i)}, \hat{\mathbf{z}}, \hat{\mathbf{y}})$ 
8:   until tired
9:   return  $\theta$ 

```

tations for many natural language sentences. For example, in the geography domain, the denotation of a question would be its answer (Clarke et al., 2010; Liang et al., 2013):

Text :*What states border Georgia?*
Logical form : $\lambda x.\text{STATE}(x) \wedge \text{BORDER}(x, \text{GEORGIA})$
Denotation :{Alabama, Florida, North Carolina,
South Carolina, Tennessee}

6436 Similarly, in a robotic control setting, the denotation of a command would be an action or
6437 sequence of actions (Artzi and Zettlemoyer, 2013). In both cases, the idea is to reward the
6438 semantic parser for choosing an analysis whose denotation is correct: the right answer to
6439 the question, or the right action.

Learning from logical forms was made possible by summing or maxing over derivations. This idea can be carried one step further, summing or maxing over all logical forms with the correct denotation. Let $v_i(\mathbf{y}) \in \{0, 1\}$ be a **validation function**, which assigns a binary score indicating whether the denotation $[\mathbf{y}]$ for the text $\mathbf{w}^{(i)}$ is correct. We can then learn by maximizing a conditional-likelihood objective,

$$\ell^{(i)}(\boldsymbol{\theta}) = \log \sum_{\mathbf{y}} v_i(\mathbf{y}) \times p(\mathbf{y} \mid \mathbf{w}; \boldsymbol{\theta}) \quad [12.29]$$

$$= \log \sum_{\mathbf{y}} v_i(\mathbf{y}) \times \sum_{\mathbf{z}} p(\mathbf{y}, \mathbf{z} \mid \mathbf{w}; \boldsymbol{\theta}), \quad [12.30]$$

6440 which sums over all derivations \mathbf{z} of all valid logical forms, $\{\mathbf{y} : v_i(\mathbf{y}) = 1\}$. This cor-
6441 responds to the log-probability that the semantic parser produces a logical form with a
6442 valid denotation.

Differentiating with respect to θ , we obtain,

$$\frac{\partial \ell^{(i)}}{\partial \theta} = \sum_{\mathbf{y}, \mathbf{z}: v_i(\mathbf{y})=1} p(\mathbf{y}, \mathbf{z} | \mathbf{w}) \mathbf{f}(\mathbf{w}, \mathbf{z}, \mathbf{y}) - \sum_{\mathbf{y}', \mathbf{z}'} p(\mathbf{y}', \mathbf{z}' | \mathbf{w}) \mathbf{f}(\mathbf{w}, \mathbf{z}', \mathbf{y}'), \quad [12.31]$$

which is the usual difference in feature expectations. The positive term computes the expected feature expectations conditioned on the denotation being valid, while the second term computes the expected feature expectations according to the current model, without regard to the ground truth. Large-margin learning formulations are also possible for this problem. For example, Artzi and Zettlemoyer (2013) generate a set of valid and invalid derivations, and then impose a constraint that all valid derivations should score higher than all invalid derivations. This constraint drives a perceptron-like learning rule.

Additional resources

A key issue not considered here is how to handle **semantic underspecification**: cases in which there are multiple semantic interpretations for a single syntactic structure. Quantifier scope ambiguity is a classic example. Blackburn and Bos (2005) enumerate a number of approaches to this issue, and also provide links between natural language semantics and computational inference techniques. Much of the contemporary research on semantic parsing uses the framework of combinatory categorial grammar (CCG). Carpenter (1997) provides a comprehensive treatment of how CCG can support compositional semantic analysis. Another recent area of research is the semantics of multi-sentence texts. This can be handled with models of **dynamic semantics**, such as dynamic predicate logic (Groenendijk and Stokhof, 1991).

Alternative readings on formal semantics include an “informal” reading from Levy and Manning (2009), and a more involved introduction from Briscoe (2011). To learn more about ongoing research on data-driven semantic parsing, readers may consult the survey article by Liang and Potts (2015), tutorial slides and videos by Artzi and Zettlemoyer (2013),¹² and the source code by Yoav Artzi¹³ and Percy Liang.¹⁴

Exercises

- Derive the **modus ponens** inference rule, which states that if we know $\phi \Rightarrow \psi$ and ϕ , then ψ must be true. The derivation can be performed using the definition of the \Rightarrow operator and some of the laws provided in § 12.2.1, plus one additional identity: $\perp \vee \phi = \phi$.

¹²Videos are currently available at <http://yoavartzi.com/tutorial/>

¹³<http://yoavartzi.com/spf>

¹⁴<https://github.com/percyliang/sempre>

- 6471 2. Convert the following examples into first-order logic, using the relations CAN-SLEEP,
 6472 MAKES-NOISE, and BROTHER.
- 6473 • If Abigail makes noise, no one can sleep.
 6474 • If Abigail makes noise, someone cannot sleep.
 6475 • None of Abigail's brothers can sleep.
 6476 • If one of Abigail's brothers makes noise, Abigail cannot sleep.
- 6477 3. Extend the grammar fragment G_1 to include the ditransitive verb *teaches* and the
 6478 proper noun *Swahili*. Show how to derive the interpretation for the sentence *Alex*
 6479 *teaches Brit Swahili*, which should be $\text{TEACHES}(\text{ALEX}, \text{BRIT}, \text{SWAHILI})$. The grammar
 6480 need not be in Chomsky Normal Form. For the ditransitive verb, use NP_1 and NP_2
 6481 to indicate the two direct objects.
- 6482 4. Derive the semantic interpretation for the sentence *Alex likes every dog*, using gram-
 6483 mar fragment G_2 .
- 6484 5. Extend the grammar fragment G_2 to handle adjectives, so that the meaning of *an
 6485 angry dog* is $\lambda P. \exists x \text{DOG}(x) \wedge \text{ANGRY}(x) \wedge P(x)$. Specifically, you should supply the
 6486 lexical entry for the adjective *angry*, and you should specify the syntactic-semantic
 6487 productions $\text{NP} \rightarrow \text{DET } \text{NOM}$, $\text{NOM} \rightarrow \text{JJ } \text{NOM}$, and $\text{NOM} \rightarrow \text{NN}$.
- 6488 6. Extend your answer to the previous question to cover copula constructions with
 6489 predicative adjectives, such as *Alex is angry*. The interpretation should be $\text{ANGRY}(\text{ALEX})$.
 6490 You should add a verb phrase production $\text{VP} \rightarrow \text{V}_{\text{cop}} \text{ JJ}$, and a terminal production
 6491 $\text{V}_{\text{cop}} \rightarrow \text{is}$. Show why your grammar extensions result in the correct interpretation.
- 6492 7. In Figure 12.6 and Figure 12.7, we treat the plurals *shoots* and *leaves* as entities. Revise
 6493 G_2 so that the interpretation of *Alex eats leaves* is $\forall x. (\text{LEAF}(x) \Rightarrow \text{EATS}(\text{ALEX}, x))$, and
 6494 show the resulting perceptron update.
- 6495 8. Statements like *every student eats a pizza* have two possible interpretations, depend-
 6496 ing on quantifier scope:

$$\forall x \exists y \text{PIZZA}(y) \wedge (\text{STUDENT}(x) \Rightarrow \text{EATS}(x, y)) \quad [12.32]$$

$$\exists y \forall x \text{PIZZA}(y) \wedge (\text{STUDENT}(x) \Rightarrow \text{EATS}(x, y)) \quad [12.33]$$

6495 Explain why these interpretations really are different, and modify the grammar G_2
 6496 so that it can produce both interpretations.

- 6497 9. Derive Equation 12.27.
- 6498 10. In the GeoQuery domain, give a natural language query that has multiple plausible
 6499 semantic interpretations with the same denotation. List both interpretaions and the
 6500 denotation.

6501 **Hint:** There are many ways to do this, but one approach involves using toponyms
6502 (place names) that could plausibly map to several different entities in the model.

6503

Chapter 13

6504

Predicate-argument semantics

6505 This chapter considers more “lightweight” semantic representations, which discard some
6506 aspects of first-order logic, but focus on predicate-argument structures. Let’s begin by
6507 thinking about the semantics of events, with a simple example:

6508 (13.1) Asha gives Boyang a book.

6509 A first-order logical representation of this sentence is,

$$\exists x. \text{BOOK}(x) \wedge \text{GIVE}(\text{ASHA}, \text{BOYANG}, x) \quad [13.1]$$

6510 In this representation, we define variable x for the book, and we link the strings *Asha* and
6511 *Boyang* to entities ASHA and BOYANG. Because the action of giving involves a giver, a
6512 recipient, and a gift, the predicate GIVE must take three arguments.

6513 Now suppose we have additional information about the event:

6514 (13.2) Yesterday, Asha reluctantly gave Boyang a book.

6515 One possible solution is to extend the predicate GIVE to take additional arguments,

$$\exists x. \text{BOOK}(x) \wedge \text{GIVE}(\text{ASHA}, \text{BOYANG}, x, \text{YESTERDAY}, \text{RELUCTANTLY}) \quad [13.2]$$

But this is clearly unsatisfactory: *yesterday* and *reluctantly* are optional arguments, and we would need a different version of the GIVE predicate for every possible combination of arguments. **Event semantics** solves this problem by **reifying** the event as an existentially quantified variable e ,

$$\begin{aligned} \exists e, x. & \text{GIVE-EVENT}(e) \wedge \text{GIVER}(e, \text{ASHA}) \wedge \text{GIFT}(e, x) \wedge \text{BOOK}(e, x) \wedge \text{RECIPIENT}(e, \text{BOYANG}) \\ & \wedge \text{TIME}(e, \text{YESTERDAY}) \wedge \text{MANNER}(e, \text{RELUCTANTLY}) \end{aligned}$$

6516 In this way, each argument of the event — the giver, the recipient, the gift — can be rep-
 6517 resented with a relation of its own, linking the argument to the event e . The expression
 6518 GIVER(e , ASHA) says that ASHA plays the **role** of GIVER in the event. This reformulation
 6519 handles the problem of optional information such as the time or manner of the event,
 6520 which are called **adjuncts**. Unlike arguments, adjuncts are not a mandatory part of the
 6521 relation, but under this representation, they can be expressed with additional logical rela-
 6522 tions that are conjoined to the semantic interpretation of the sentence.¹

6523 The event semantic representation can be applied to nested clauses, e.g.,

6524 (13.3) Chris sees Asha pay Boyang.

This is done by using the event variable as an argument:

$$\begin{aligned} \exists e_1 \exists e_2 \text{SEE-EVENT}(e_1) \wedge \text{SEER}(e_1, \text{CHRIS}) \wedge \text{SIGHT}(e_1, e_2) \\ \wedge \text{PAY-EVENT}(e_2) \wedge \text{PAYER}(e_2, \text{ASHA}) \wedge \text{PAYEE}(e_2, \text{BOYANG}) \end{aligned} \quad [13.3]$$

6525 As with first-order logic, the goal of event semantics is to provide a representation that
 6526 generalizes over many surface forms. Consider the following paraphrases of (13.1):

- 6527 (13.4) Asha gives a book to Boyang.
- 6528 (13.5) A book is given to Boyang by Asha.
- 6529 (13.6) A book is given by Asha to Boyang.
- 6530 (13.7) The gift of a book from Asha to Boyang ...

6531 All have the same event semantic meaning as Equation 13.1, but the ways in which the
 6532 meaning can be expressed are diverse. The final example does not even include a verb:
 6533 events are often introduced by verbs, but as shown by (13.7), the noun *gift* can introduce
 6534 the same predicate, with the same accompanying arguments.

6535 **Semantic role labeling** (SRL) is a relaxed form of semantic parsing, in which each
 6536 semantic role is filled by a set of tokens from the text itself. This is sometimes called
 6537 “shallow semantics” because, unlike model-theoretic semantic parsing, role fillers need
 6538 not be symbolic expressions with denotations in some world model. A semantic role
 6539 labeling system is required to identify all predicates, and then specify the spans of text
 6540 that fill each role. To give a sense of the task, here is a more complicated example:

- 6541 (13.8) Boyang wants Asha to give him a linguistics book.

¹This representation is often called **Neo-Davidsonian event semantics**. The use of existentially-quantified event variables was proposed by Davidson (1967) to handle the issue of optional adjuncts. In Neo-Davidsonian semantics, this treatment of adjuncts is extended to mandatory arguments as well (e.g., Parsons, 1990).

6542 In this example, there are two predicates, expressed by the verbs *want* and *give*. Thus, a
 6543 semantic role labeler might return the following output:

- 6544 • (PREDICATE : *wants*, WANTED : *Boyang*, DESIRE : *Asha to give him a linguistics book*)
 6545 • (PREDICATE : *give*, GIVER : *Asha*, RECIPIENT : *him*, GIFT : *a linguistics book*)

6546 *Boyang* and *him* may refer to the same person, but the semantic role labeling is not re-
 6547 quired to resolve this reference. Other predicate-argument representations, such as **Ab-**
 6548 **stract Meaning Representation (AMR)**, do require reference resolution. We will return to
 6549 AMR in § 13.3, but first, let us further consider the definition of semantic roles.

6550 13.1 Semantic roles

6551 In event semantics, it is necessary to specify a number of additional logical relations to
 6552 link arguments to events: GIVER, RECIPIENT, SEER, SIGHT, etc. Indeed, every predicate re-
 6553 quires a set of logical relations to express its own arguments. In contrast, adjuncts such as
 6554 TIME and MANNER are shared across many types of events. A natural question is whether
 6555 it is possible to treat mandatory arguments more like adjuncts, by identifying a set of
 6556 generic argument types that are shared across many event predicates. This can be further
 6557 motivated by examples involving related verbs:

- 6558 (13.9) Asha gave Boyang a book.
 6559 (13.10) Asha loaned Boyang a book.
 6560 (13.11) Asha taught Boyang a lesson.
 6561 (13.12) Asha gave Boyang a lesson.

6562 The respective roles of Asha, Boyang, and the book are nearly identical across the first
 6563 two examples. The third example is slightly different, but the fourth example shows that
 6564 the roles of GIVER and TEACHER can be viewed as related.

6565 One way to think about the relationship between roles such as GIVER and TEACHER is
 6566 by enumerating the set of properties that an entity typically possesses when it fulfills these
 6567 roles: givers and teachers are usually **animate** (they are alive and sentient) and **volitional**
 6568 (they choose to enter into the action).² In contrast, the thing that gets loaned or taught is
 6569 usually not animate or volitional; furthermore, it is unchanged by the event.

6570 Building on these ideas, **thematic roles** generalize across predicates by leveraging the
 6571 shared semantic properties of typical role fillers (Fillmore, 1968). For example, in exam-
 6572 ples (13.9-13.12), Asha plays a similar role in all four sentences, which we will call the

²There are always exceptions. For example, in the sentence *The C programming language has taught me a lot about perseverance*, the “teacher” is the *The C programming language*, which is presumably not animate or volitional.

	<i>Asha</i>	<i>gave</i>	<i>Boyang</i>	<i>a book</i>
VerbNet	AGENT		RECIPIENT	THEME
PropBank	ARG0: giver		ARG2: entity given to	ARG1: thing given
FrameNet	DONOR		RECIPIENT	THEME
	<i>Asha</i>	<i>taught</i>	<i>Boyang</i>	<i>algebra</i>
VerbNet	AGENT		RECIPIENT	TOPIC
PropBank	ARG0: teacher		ARG2: student	ARG1: subject
FrameNet	TEACHER		STUDENT	SUBJECT

Figure 13.1: Example semantic annotations according to VerbNet, PropBank, and FrameNet

6573 **agent.** This reflects several shared semantic properties: she is the one who is actively and
 6574 intentionally performing the action, while Boyang is a more passive participant; the book
 6575 and the lesson would play a different role, as non-animate participants in the event.

6576 Example annotations from three well known systems are shown in Figure 13.1. We
 6577 will now discuss these systems in more detail.

6578 13.1.1 VerbNet

6579 **VerbNet** (Kipper-Schuler, 2005) is a lexicon of verbs, and it includes thirty “core” thematic
 6580 roles played by arguments to these verbs. Here are some example roles, accompanied by
 6581 their definitions from the VerbNet Guidelines.³

- 6582 • AGENT: “ACTOR in an event who initiates and carries out the event intentionally or
 6583 consciously, and who exists independently of the event.”
- 6584 • PATIENT: “UNDERGOER in an event that experiences a change of state, location or
 6585 condition, that is causally involved or directly affected by other participants, and
 6586 exists independently of the event.”
- 6587 • RECIPIENT: “DESTINATION that is animate”
- 6588 • THEME: “UNDERGOER that is central to an event or state that does not have control
 6589 over the way the event occurs, is not structurally changed by the event, and/or is
 6590 characterized as being in a certain position or condition throughout the state.”
- 6591 • TOPIC: “THEME characterized by information content transferred to another partic-
 6592 ipant.”

³http://verbs.colorado.edu/verb-index/VerbNet_Guidelines.pdf

6593 VerbNet roles are organized in a hierarchy, so that a TOPIC is a type of THEME, which in
 6594 turn is a type of UNDERGOER, which is a type of PARTICIPANT, the top-level category.

6595 In addition, VerbNet organizes verb senses into a class hierarchy, in which verb senses
 6596 that have similar meanings are grouped together. Recall from § 4.2 that multiple meanings
 6597 of the same word are called **senses**, and that WordNet identifies senses for many English
 6598 words. VerbNet builds on WordNet, so that verb classes are identified by the WordNet
 6599 senses of the verbs that they contain. For example, the verb class give-13.1 includes
 6600 the first WordNet sense of *loan* and the second WordNet sense of *lend*.

6601 Each VerbNet class or subclass takes a set of thematic roles. For example, give-13.1
 6602 takes arguments with the thematic roles of AGENT, THEME, and RECIPIENT;⁴ the pred-
 6603 icate TEACH takes arguments with the thematic roles AGENT, TOPIC, RECIPIENT, and
 6604 SOURCE.⁵ So according to VerbNet, *Asha* and *Boyang* play the roles of AGENT and RECIP-
 6605 IENT in the sentences,

6606 (13.13) Asha gave Boyang a book.

6607 (13.14) Asha taught Boyang algebra.

6608 The *book* and *algebra* are both THEMES, but *algebra* is a subcategory of THEME — a TOPIC
 6609 — because it consists of information content that is given to the receiver.

6610 13.1.2 Proto-roles and PropBank

6611 Detailed thematic role inventories of the sort used in VerbNet are not universally accepted.
 6612 For example, Dowty (1991, pp. 547) notes that “Linguists have often found it hard to agree
 6613 on, and to motivate, the location of the boundary between role types.” He argues that a
 6614 solid distinction can be identified between just two **proto-roles**:

6615 **Proto-Agent.** Characterized by volitional involvement in the event or state; sentience
 6616 and/or perception; causing an event or change of state in another participant; move-
 6617 ment; exists independently of the event.

6618 **Proto-Patient.** Undergoes change of state; causally affected by another participant; sta-
 6619 tionary relative to the movement of another participant; does not exist indepen-
 6620 dently of the event.⁶

⁴<https://verbs.colorado.edu/verb-index/vn/give-13.1.php>

⁵https://verbs.colorado.edu/verb-index/vn/transfer_mesg-37.1.1.php

⁶Reisinger et al. (2015) ask crowd workers to annotate these properties directly, finding that annotators tend to agree on the properties of each argument. They also find that in English, arguments having more proto-agent properties tend to appear in subject position, while arguments with more proto-patient properties appear in object position.

6621 In the examples in Figure 13.1, Asha has most of the proto-agent properties: in giving
 6622 the book to Boyang, she is acting volitionally (as opposed to *Boyang got a book from Asha*, in
 6623 which it is not clear whether Asha gave up the book willingly); she is sentient; she causes
 6624 a change of state in Boyang; she exists independently of the event. Boyang has some
 6625 proto-agent properties: he is sentient and exists independently of the event. But he also
 6626 some proto-patient properties: he is the one who is causally affected and who undergoes
 6627 change of state. The book that Asha gives Boyang has even fewer of the proto-agent
 6628 properties: it is not volitional or sentient, and it has no causal role. But it also lacks many
 6629 of the proto-patient properties: it does not undergo change of state, exists independently
 6630 of the event, and is not stationary.

6631 The **Proposition Bank**, or PropBank (Palmer et al., 2005), builds on this basic agent-
 6632 patient distinction, as a middle ground between generic thematic roles and roles that are
 6633 specific to each predicate. Each verb is linked to a list of numbered arguments, with ARG0
 6634 as the proto-agent and ARG1 as the proto-patient. Additional numbered arguments are
 6635 verb-specific. For example, for the predicate TEACH,⁷ the arguments are:

- 6636 • ARG0: the teacher
- 6637 • ARG1: the subject
- 6638 • ARG2: the student(s)

6639 Verbs may have any number of arguments: for example, WANT and GET have five, while
 6640 EAT has only ARG0 and ARG1. In addition to the semantic arguments found in the frame
 6641 files, roughly a dozen general-purpose **adjuncts** may be used in combination with any
 6642 verb. These are shown in Table 13.1.

6643 PropBank-style semantic role labeling is annotated over the entire Penn Treebank. This
 6644 annotation includes the sense of each verbal predicate, as well as the argument spans.

6645 13.1.3 FrameNet

6646 Semantic **frames** are descriptions of situations or events. Frames may be *evoked* by one
 6647 of their **lexical units** (often a verb, but not always), and they include some number of
 6648 **frame elements**, which are like roles (Fillmore, 1976). For example, the act of teaching
 6649 is a frame, and can be evoked by the verb *taught*; the associated frame elements include
 6650 the teacher, the student(s), and the subject being taught. Frame semantics has played a
 6651 significant role in the history of artificial intelligence, in the work of Minsky (1974) and
 6652 Schank and Abelson (1977). In natural language processing, the theory of frame semantics
 6653 has been implemented in **FrameNet** (Fillmore and Baker, 2009), which consists of a lexicon

⁷<http://verbs.colorado.edu/propbank/framesets-english-aliases/teach.html>

TMP	time	<i>Boyang ate a bagel</i> [AM-TMP <i>yesterday</i>].
LOC	location	<i>Asha studies in</i> [AM-LOC <i>Stuttgart</i>]
MOD	modal verb	<i>Asha</i> [AM-MOD <i>will</i>] <i>study in Stuttgart</i>
ADV	general purpose	[AM-ADV <i>Luckily</i>], <i>Asha knew algebra</i> .
MNR	manner	<i>Asha ate</i> [AM-MNR <i>aggressively</i>].
DIS	discourse connective	[AM-DIS <i>However</i>], <i>Asha prefers algebra</i> .
PRP	purpose	<i>Barry studied</i> [AM-PRP <i>to pass the bar</i>].
DIR	direction	<i>Workers dumped burlap sacks</i> [AM-DIR <i>into a bin</i>].
NEG	negation	<i>Asha does</i> [AM-NEG <i>not</i>] <i>speak Albanian</i> .
EXT	extent	<i>Prices increased</i> [AM-EXT <i>4%</i>].
CAU	cause	<i>Boyang returned the book</i> [AM-CAU <i>because it was overdue</i>].

Table 13.1: PropBank adjuncts (Palmer et al., 2005), sorted by frequency in the corpus

6654 of roughly 1000 frames, and a corpus of more than 200,000 “exemplar sentences,” in which
 6655 the frames and their elements are annotated.⁸

6656 Rather than seeking to link semantic roles such as TEACHER and GIVER into the-
 6657 matic roles such as AGENT, FrameNet aggressively groups verbs into frames, and links
 6658 semantically-related roles across frames. For example, the following two sentences would
 6659 be annotated identically in FrameNet:

6660 (13.15) Asha taught Boyang algebra.

6661 (13.16) Boyang learned algebra from Asha.

6662 This is because *teach* and *learn* are both lexical units in the EDUCATION-TEACHING frame.
 6663 Furthermore, roles can be shared even when the frames are distinct, as in the following
 6664 two examples:

6665 (13.17) Asha gave Boyang a book.

6666 (13.18) Boyang got a book from Asha.

6667 The GIVING and GETTING frames both have RECIPIENT and THEME elements, so Boyang
 6668 and the book would play the same role. Asha’s role is different: she is the DONOR in the
 6669 GIVING frame, and the SOURCE in the GETTING frame. FrameNet makes extensive use of
 6670 multiple inheritance to share information across frames and frame elements: for example,
 6671 the COMMERCE-SELL and LENDING frames inherit from GIVING frame.

⁸Current details and data can be found at <https://framenet.icsi.berkeley.edu/>

6672 13.2 Semantic role labeling

6673 The task of semantic role labeling is to identify the parts of the sentence comprising the
 6674 semantic roles. In English, this task is typically performed on the PropBank corpus, with
 6675 the goal of producing outputs in the following form:

6676 (13.19) [ARG0 Asha] [GIVE.01 gave] [ARG2 Boyang's mom] [ARG1 a book] [AM-TMP yesterday].

6677 Note that a single sentence may have multiple verbs, and therefore a given word may be
 6678 part of multiple role-fillers:

6679 (13.20) [ARG0 Asha] [WANT.01 wanted]
 Asha wanted

6680 [ARG1 Boyang to give her the book].
 [ARG0 Boyang] [GIVE.01 to give] [ARG2 her] [ARG1 the book].

6681 13.2.1 Semantic role labeling as classification

6682 PropBank is annotated on the Penn Treebank, and annotators used phrasal constituents
 6683 (\S 9.2.2) to fill the roles. PropBank semantic role labeling can be viewed as the task of as-
 6684 signing to each phrase a label from the set $\mathcal{R} = \{\emptyset, \text{PRED}, \text{ARG0}, \text{ARG1}, \text{ARG2}, \dots, \text{AM-LOC}, \text{AM-TMP}, \dots\}$
 6685 with respect to each predicate. If we treat semantic role labeling as a classification prob-
 6686 lem, we obtain the following functional form:

$$\hat{y}_{(i,j)} = \underset{y}{\operatorname{argmax}} \psi(\mathbf{w}, y, i, j, \rho, \tau), \quad [13.4]$$

6687 where,

- 6688 • (i, j) indicates the span of a phrasal constituent $(w_{i+1}, w_{i+2}, \dots, w_j)$;⁹
- 6689 • \mathbf{w} represents the sentence as a sequence of tokens;
- 6690 • ρ is the index of the predicate verb in \mathbf{w} ;
- 6691 • τ is the structure of the phrasal constituent parse of \mathbf{w} .

6692 Early work on semantic role labeling focused on discriminative feature-based models,
 6693 where $\psi(\mathbf{w}, y, i, j, \rho, \tau) = \theta \cdot f(\mathbf{w}, y, i, j, \rho, \tau)$. Table 13.2 shows the features used in a sem-
 6694 inal paper on FrameNet semantic role labeling (Gildea and Jurafsky, 2002). By 2005 there

⁹PropBank roles can also be filled by **split constituents**, which are discontinuous spans of text. This situation most frequently in reported speech, e.g. [ARG1 *By addressing these problems*], *Mr. Maxwell said*, [ARG1 *the new funds have become extremely attractive.*] (example adapted from Palmer et al., 2005). This issue is typically addressed by defining “continuation arguments”, e.g. C-ARG1, which refers to the continuation of ARG1 after the split.

Predicate lemma and POS tag	The lemma of the predicate verb and its part-of-speech tag
Voice	Whether the predicate is in active or passive voice, as determined by a set of syntactic patterns for identifying passive voice constructions
Phrase type	The constituent phrase type for the proposed argument in the parse tree, e.g. NP, PP
Headword and POS tag	The head word of the proposed argument and its POS tag, identified using the Collins (1997) rules
Position	Whether the proposed argument comes before or after the predicate in the sentence
Syntactic path	The set of steps on the parse tree from the proposed argument to the predicate (described in detail in the text)
Subcategorization	The syntactic production from the first branching node above the predicate. For example, in Figure 13.2, the subcategorization feature around <i>taught</i> would be VP → VBD NP PP.

Table 13.2: Features used in semantic role labeling by Gildea and Jurafsky (2002).

6695 were several systems for PropBank semantic role labeling, and their approaches and fea-
 6696 ture sets are summarized by Carreras and Márquez (2005). Typical features include: the
 6697 phrase type, head word, part-of-speech, boundaries, and neighbors of the proposed argu-
 6698 ment $w_{i+1:j}$; the word, lemma, part-of-speech, and voice of the verb w_ρ (active or passive),
 6699 as well as features relating to its frameset; the distance and path between the verb and
 6700 the proposed argument. In this way, semantic role labeling systems are high-level “con-
 6701 sumers” in the NLP stack, using features produced from lower-level components such as
 6702 part-of-speech taggers and parsers. More comprehensive feature sets are enumerated by
 6703 Das et al. (2014) and Täckström et al. (2015).

6704 A particularly powerful class of features relate to the **syntactic path** between the ar-
 6705 gument and the predicate. These features capture the sequence of moves required to get
 6706 from the argument to the verb by traversing the phrasal constituent parse of the sentence.
 6707 The idea of these features is to capture syntactic regularities in how various arguments
 6708 are realized. Syntactic path features are best illustrated by example, using the parse tree
 6709 in Figure 13.2:

- 6710 • The path from *Asha* to the verb *taught* is NNP↑NP↑S↓VP↓VBD. The first part of
 6711 the path, NNP↑NP↑S, means that we must travel up the parse tree from the NNP
 6712 tag (proper noun) to the S (sentence) constituent. The second part of the path,
 6713 S↓VP↓VBD, means that we reach the verb by producing a VP (verb phrase) from

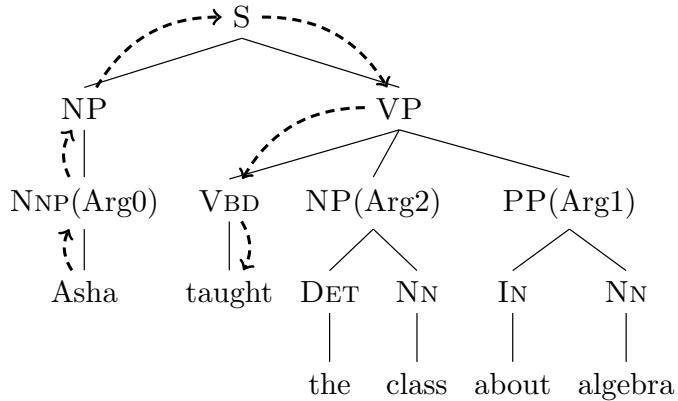


Figure 13.2: Semantic role labeling on the phrase-structure parse tree for a sentence. The dashed line indicates the syntactic path from *Asha* to the predicate verb *taught*.

6714 the S constituent, and then by producing a VBD (past tense verb). This feature is
 6715 consistent with *Asha* being in subject position, since the path includes the sentence
 6716 root S.

- 6717 • The path from *the class* to *taught* is NP↑VP↓VBD. This is consistent with *the class*
 6718 being in object position, since the path passes through the VP node that dominates
 6719 the verb *taught*.

6720 Because there are many possible path features, it can also be helpful to look at smaller
 6721 parts: for example, the upward and downward parts can be treated as separate features;
 6722 another feature might consider whether S appears anywhere in the path.

6723 Rather than using the constituent parse, it is also possible to build features from the
 6724 **dependency path** between the head word of each argument and the verb (Pradhan et al.,
 6725 2005). Using the Universal Dependency part-of-speech tagset and dependency relations (Nivre
 6726 et al., 2016), the dependency path from *Asha* to *taught* is PROPN $\xleftarrow[\text{NSUBJ}]{} \text{VERB}$, because *taught*
 6727 is the head of a relation of type $\xleftarrow[\text{NSUBJ}]{} \text{VERB}$ with *Asha*. Similarly, the dependency path from *class*
 6728 to *taught* is NOUN $\xleftarrow[\text{DOBJ}]{} \text{VERB}$, because *class* heads the noun phrase that is a direct object of
 6729 *taught*. A more interesting example is *Asha wanted to teach the class*, where the path from
 6730 *Asha* to *teach* is PROPN $\xleftarrow[\text{NSUBJ}]{} \text{VERB} \rightarrow[\text{XCOMP}] \text{VERB}$. The right-facing arrow in second relation
 6731 indicates that *wanted* is the head of its XCOMP relation with *teach*.

6732 **13.2.2 Semantic role labeling as constrained optimization**

6733 A potential problem with treating SRL as a classification problem is that there are a num-
6734 ber of sentence-level **constraints**, which a classifier might violate.

- 6735 • For a given verb, there can be only one argument of each type (ARG0, ARG1, etc.)
 6736 • Arguments cannot overlap. This problem arises when we are labeling the phrases
 6737 in a constituent parse tree, as shown in Figure 13.2: if we label the PP *about algebra*
 6738 as an argument or adjunct, then its children *about* and *algebra* must be labeled as \emptyset .
 6739 The same constraint also applies to the syntactic ancestors of this phrase.

6740 These constraints introduce dependencies across labeling decisions. In structure pre-
 6741 diction problems such as sequence labeling and parsing, such dependencies are usually
 6742 handled by defining a scoring over the entire structure, \mathbf{y} . Efficient inference requires
 6743 that the global score decomposes into local parts: for example, in sequence labeling, the
 6744 scoring function decomposes into scores of pairs of adjacent tags, permitting the applica-
 6745 tion of the Viterbi algorithm for inference. But the constraints that arise in semantic role
 6746 labeling are less amenable to local decomposition.¹⁰ We therefore consider **constrained**
 6747 **optimization** as an alternative solution.

Let the set $\mathcal{C}(\tau)$ refer to all labelings that obey the constraints introduced by the parse τ . The semantic role labeling problem can be reformulated as a constrained optimization over $\mathbf{y} \in \mathcal{C}(\tau)$,

$$\begin{aligned} \max_{\mathbf{y}} \quad & \sum_{(i,j) \in \tau} \psi(\mathbf{w}, y_{i,j}, i, j, \rho, \tau) \\ \text{s.t. } \quad & \mathbf{y} \in \mathcal{C}(\tau). \end{aligned} \quad [13.5]$$

6748 In this formulation, the objective (shown on the first line) is a separable function of each
 6749 individual labeling decision, but the constraints (shown on the second line) apply to the
 6750 overall labeling. The sum $\sum_{(i,j) \in \tau}$ indicates that we are summing over all constituent
 6751 spans in the parse τ . The expression s.t. in the second line means that we maximize the
 6752 objective *subject to* the constraint $\mathbf{y} \in \mathcal{C}(\tau)$.

6753 A number of practical algorithms exist for restricted forms of constrained optimiza-
 6754 tion. One such restricted form is **integer linear programming**, in which the objective and
 6755 constraints are linear functions of integer variables. To formulate SRL as an integer linear
 6756 program, we begin by rewriting the labels as a set of binary variables $\mathbf{z} = \{z_{i,j,r}\}$ (Pun-
 6757 yakanok et al., 2008),

$$z_{i,j,r} = \begin{cases} 1, & y_{i,j} = r \\ 0, & \text{otherwise,} \end{cases} \quad [13.6]$$

¹⁰Dynamic programming solutions have been proposed by Tromble and Eisner (2006) and Täckström et al. (2015), but they involve creating a trellis structure whose size is exponential in the number of labels.

6758 where $r \in \mathcal{R}$ is a label in the set $\{\text{ARG0}, \text{ARG1}, \dots, \text{AM-LOC}, \dots, \emptyset\}$. Thus, the variables
 6759 \mathbf{z} are a binarized version of the semantic role labeling \mathbf{y} .

The objective can then be formulated as a linear function of \mathbf{z} .

$$\sum_{(i,j) \in \tau} \psi(\mathbf{w}, y_{i,j}, i, j, \rho, \tau) = \sum_{i,j,r} \psi(\mathbf{w}, r, i, j, \rho, \tau) \times z_{i,j,r}, \quad [13.7]$$

6760 which is the sum of the scores of all relations, as indicated by $z_{i,j,r}$.

Constraints Integer linear programming permits linear inequality constraints, of the general form $\mathbf{A}\mathbf{z} \leq \mathbf{b}$, where the parameters \mathbf{A} and \mathbf{b} define the constraints. To make this more concrete, let's start with the constraint that each non-null role type can occur only once in a sentence. This constraint can be written,

$$\forall r \neq \emptyset, \quad \sum_{(i,j) \in \tau} z_{i,j,r} \leq 1. \quad [13.8]$$

6761 Recall that $z_{i,j,r} = 1$ iff the span (i, j) has label r ; this constraint says that for each possible
 6762 label $r \neq \emptyset$, there can be at most one (i, j) such that $z_{i,j,r} = 1$. Rewriting this constraint
 6763 can be written in the form $\mathbf{A}\mathbf{z} \leq \mathbf{b}$, as you will find if you complete the exercises at the
 6764 end of the chapter.

Now consider the constraint that labels cannot overlap. Let's define the convenience function $o((i, j), (i', j')) = 1$ iff (i, j) overlaps (i', j') , and zero otherwise. Thus, o will indicate if a constituent (i', j') is either an ancestor or descendant of (i, j) . The constraint is that if two constituents overlap, only one can have a non-null label:

$$\forall (i, j) \in \tau, \quad \sum_{(i', j') \in \tau} \sum_{r \neq \emptyset} o((i, j), (i', j')) \times z_{i',j',r} \leq 1, \quad [13.9]$$

6765 where $o((i, j), (i, j)) = 1$.

In summary, the semantic role labeling problem can thus be rewritten as the following integer linear program,

$$\max_{\mathbf{z} \in \{0,1\}^{|\tau|}} \quad \sum_{(i,j) \in \tau} \sum_{r \in \mathcal{R}} z_{i,j,r} \psi_{i,j,r} \quad [13.10]$$

$$s.t. \quad \forall r \neq \emptyset, \quad \sum_{(i,j) \in \tau} z_{i,j,r} \leq 1. \quad [13.11]$$

$$\forall (i, j) \in \tau, \quad \sum_{(i', j') \in \tau} \sum_{r \neq \emptyset} o((i, j), (i', j')) \times z_{i',j',r} \leq 1. \quad [13.12]$$

6766 **Learning with constraints** Learning can be performed in the context of constrained op-
 6767 timization using the usual perceptron or large-margin classification updates. Because
 6768 constrained inference is generally more time-consuming, a key question is whether it is
 6769 necessary to apply the constraints during learning. Chang et al. (2008) find that better per-
 6770 formance can be obtained by learning *without* constraints, and then applying constraints
 6771 only when using the trained model to predict semantic roles for unseen data.

6772 **How important are the constraints?** Das et al. (2014) find that an unconstrained, classification-
 6773 based method performs nearly as well as constrained optimization for FrameNet parsing;
 6774 while it commits many violations of the “no-overlap” constraint, the overall F_1 score is
 6775 less than one point worse than the score at the constrained optimum. Similar results
 6776 were obtained for PropBank semantic role labeling by Punyakanok et al. (2008). He et al.
 6777 (2017) find that constrained inference makes a bigger impact if the constraints are based
 6778 on manually-labeled “gold” syntactic parses. This implies that errors from the syntac-
 6779 tic parser may limit the effectiveness of the constraints. Punyakanok et al. (2008) hedge
 6780 against parser error by including constituents from several different parsers; any con-
 6781 stituent can be selected from any parse, and additional constraints ensure that overlap-
 6782 ping constituents are not selected.

6783 **Implementation** Integer linear programming solvers such as `glpk`,¹¹ `cplex`,¹² and `Gurobi`¹³
 6784 allow inequality constraints to be expressed directly in the problem definition, rather than
 6785 in the matrix form $\mathbf{A}z \leq b$. The time complexity of integer linear programming is theoreti-
 6786 cally exponential in the number of variables $|z|$, but in practice these off-the-shelf solvers
 6787 obtain good solutions efficiently. Das et al. (2014) report that the `cplex` solver requires 43
 6788 seconds to perform inference on the FrameNet test set, which contains 4,458 predicates.

6789 Recent work has shown that many constrained optimization problems in natural lan-
 6790 guage processing can be solved in a highly parallelized fashion, using optimization tech-
 6791 niques such as **dual decomposition**, which are capable of exploiting the underlying prob-
 6792 lem structure (Rush et al., 2010). Das et al. (2014) apply this technique to FrameNet se-
 6793 mantic role labeling, obtaining an order-of-magnitude speedup over `cplex`.

6794 13.2.3 Neural semantic role labeling

6795 Neural network approaches to SRL have tended to treat it as a sequence labeling task,
 6796 using a labeling scheme such as the **BIO notation**, which we previously saw in named
 6797 entity recognition (§ 8.3). In this notation, the first token in a span of type ARG1 is labeled

¹¹<https://www.gnu.org/software/glpk/>

¹²<https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>

¹³<http://www.gurobi.com/>

6798 B-ARG1; all remaining tokens in the span are *inside*, and are therefore labeled I-ARG1.
 6799 Tokens outside any argument are labeled O. For example:

- 6800 (13.21) *Asha taught Boyang 's mom about algebra*
 B-ARG0 PRED B-ARG2 I-ARG2 I-ARG2 B-ARG1 I-ARG1

Recurrent neural networks are a natural approach to this tagging task. For example, Zhou and Xu (2015) apply a deep bidirectional multilayer LSTM (see § 7.6) to PropBank semantic role labeling. In this model, each bidirectional LSTM serves as input for another, higher-level bidirectional LSTM, allowing complex non-linear transformations of the original input embeddings, $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_M]$. The hidden state of the final LSTM is $\mathbf{Z}^{(K)} = [\mathbf{z}_1^{(K)}, \mathbf{z}_2^{(K)}, \dots, \mathbf{z}_M^{(K)}]$. The “emission” score for each tag $Y_m = y$ is equal to the inner product $\theta_y \cdot \mathbf{z}_m^{(K)}$, and there is also a transition score for each pair of adjacent tags. The complete model can be written,

$$\mathbf{Z}^{(1)} = \text{BiLSTM}(\mathbf{X}) \quad [13.13]$$

$$\mathbf{Z}^{(i)} = \text{BiLSTM}(\mathbf{Z}^{(i-1)}) \quad [13.14]$$

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}} \sum_{m=1}^M \Theta^{(y)} \mathbf{z}_m^{(K)} + \psi_{y_{m-1}, y_m}. \quad [13.15]$$

6801 Note that the final step maximizes over the entire labeling \mathbf{y} , and includes a score for
 6802 each tag transition ψ_{y_{m-1}, y_m} . This combination of LSTM and pairwise potentials on tags
 6803 is an example of an **LSTM-CRF**. The maximization over \mathbf{y} is performed by the Viterbi
 6804 algorithm.

6805 This model strongly outperformed alternative approaches at the time, including con-
 6806 strained decoding and convolutional neural networks.¹⁴ More recent work has combined
 6807 recurrent neural network models with constrained decoding, using the A^* search algo-
 6808 rithm to search over labelings that are feasible with respect to the constraints (He et al.,
 6809 2017). This yields small improvements over the method of Zhou and Xu (2015). He et al.
 6810 (2017) obtain larger improvements by creating an **ensemble** of SRL systems, each trained
 6811 on an 80% subsample of the corpus. The average prediction across this ensemble is more
 6812 robust than any individual model.

6813 13.3 Abstract Meaning Representation

6814 Semantic role labeling transforms the task of semantic parsing to a labeling task. Consider
 6815 the sentence,

¹⁴The successful application of convolutional neural networks to semantic role labeling by Collobert and Weston (2008) was an influential early result in the most recent wave of neural networks in natural language processing.

```
(w / want-01
  :ARG0 (b / boy)
  :ARG1 (g / go-02
    :ARG0 b))
```

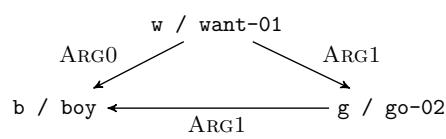


Figure 13.3: Two views of the AMR representation for the sentence *The boy wants to go.*

6816 (13.22) The boy wants to go.

6817 The PropBank semantic role labeling analysis is:

- 6818 • (PREDICATE : *wants*, ARG0 : *the boy*, ARG1 : *to go*)
- 6819 • (PREDICATE : *go*, ARG1 : *the boy*)

6820 The **Abstract Meaning Representation (AMR)** unifies this analysis into a graph structure, in which each node is a **variable**, and each edge indicates a **concept** (Banarescu et al., 2013). This can be written in two ways, as shown in Figure 13.3. On the left is the 6821 PENMAN notation (Matthiessen and Bateman, 1991), in which each set of parentheses 6822 introduces a variable. Each variable is an **instance** of a concept, which is indicated with 6823 the slash notation: for example, *w / want-01* indicates that the variable *w* is an instance 6824 of the concept *want-01*, which in turn refers to the PropBank frame for the first sense 6825 of the verb *want*. Relations are introduced with colons: for example, *:ARG0 (b / boy)* 6826 indicates a relation of type *ARG0* with the newly-introduced variable *b*. Variables can be 6827 reused, so that when the variable *b* appears again as an argument to *g*, it is understood to 6828 refer to the same boy in both cases. This arrangement is indicated compactly in the graph 6829 structure on the right, with edges indicating concepts.

6830 One way in which AMR differs from PropBank-style semantic role labeling is that it 6831 reifies each entity as a variable: for example, *the boy* in (13.22) is reified in the variable 6832 *b*, which is reused as *ARG0* in its relationship with *w / want-01*, and as *ARG1* in its 6833 relationship with *g / go-02*. Reifying entities as variables also makes it possible to 6834 represent the substructure of noun phrases more explicitly. For example, *Asha borrowed* 6835 *the algebra book* would be represented as:

```
6838 (b / borrow-01
  :ARG0 (p / person
    :name (n / name
      :op1 "Asha"))
  :ARG1 (b2 / book
    :topic (a / algebra)))
```

6844 This indicates that the variable *p* is a person, whose name is the variable *n*; that name
 6845 has one token, the string *Asha*. Similarly, the variable *b2* is a book, and the topic of *b2*
 6846 is a variable *a* whose type is algebra. The relations name and topic are examples of
 6847 **non-core roles**, which are similar to adjunct modifiers in PropBank. However, AMR’s
 6848 inventory is more extensive, including more than 70 non-core roles, such as negation,
 6849 time, manner, frequency, and location. Lists and sequences — such as the list of tokens in
 6850 a name — are described using the roles *op1*, *op2*, etc.

6851 Another feature of AMR is that a semantic predicate can be introduced by any syntac-
 6852 tic element, as in the following examples from Banarescu et al. (2013):

- 6853 (13.23) The boy destroyed the room.
- 6854 (13.24) the destruction of the room by the boy ...
- 6855 (13.25) the boy’s destruction of the room ...

6856 All these examples have the same semantics in AMR,

```
6857 (d / destroy-01
6858   :ARG0 (b / boy)
6859   :ARG1 (r / room))
```

6860 The noun *destruction* is linked to the verb *destroy*, which is captured by the PropBank
 6861 frame *destroy-01*. This can happen with adjectives as well: in the phrase *the attractive*
 6862 *spy*, the adjective *attractive* is linked to the PropBank frame *attract-01*:

```
6863 (s / spy
6864   :ARG0-of (a / attract-01))
```

6865 In this example, *ARG0-of* is an **inverse relation**, indicating that *s* is the *ARG0* of the
 6866 predicate *a*. Inverse relations make it possible for all AMR parses to have a single root
 6867 concept, which should be the **focus** of the utterance.

6868 While AMR goes farther than semantic role labeling, it does not link semantically-
 6869 related frames such as buy/sell (as FrameNet does), does not handle quantification (as
 6870 first-order predicate calculus does), and makes no attempt to handle noun number and
 6871 verb tense (as PropBank does). A recent survey by Abend and Rappoport (2017) situ-
 6872 ates AMR with respect to several other semantic representation schemes. Other linguistic
 6873 features of AMR are summarized in the original paper (Banarescu et al., 2013) and the
 6874 tutorial slides by Schneider et al. (2015).

6875 13.3.1 AMR Parsing

6876 Abstract Meaning Representation is not a labeling of the original text — unlike PropBank
6877 semantic role labeling, and most of the other tagging and parsing tasks that we have
6878 encountered thus far. The AMR for a given sentence may include multiple concepts for
6879 single words in the sentence: as we have seen, the sentence *Asha likes algebra* contains both
6880 person and name concepts for the word *Asha*. Conversely, words in the sentence may not
6881 appear in the AMR: in *Boyang made a tour of campus*, the **light verb** *make* would not appear
6882 in the AMR, which would instead be rooted on the predicate *tour*. As a result, AMR
6883 is difficult to parse, and even evaluating AMR parsing involves considerable algorithmic
6884 complexity (Cai and Yates, 2013).

6885 A further complexity is that AMR labeled datasets do not explicitly show the **alignment**
6886 between the AMR annotation and the words in the sentence. For example, the link
6887 between the word *wants* and the concept *want-01* is not annotated. To acquire training
6888 data for learning-based parsers, it is therefore necessary to first perform an alignment
6889 between the training sentences and their AMR parses. Flanigan et al. (2014) introduce a
6890 rule-based parser, which links text to concepts through a series of increasingly high-recall
6891 steps.

6892 **Graph-based parsing** One family of approaches to AMR parsing is similar to the graph-
6893 based methods that we encountered in syntactic dependency parsing (chapter 11). For
6894 these systems (Flanigan et al., 2014), parsing is a two-step process:

- 6895 1. **Concept identification** (Figure 13.4a). This involves constructing concept subgraphs
6896 for individual words or spans of adjacent words. For example, in the sentence,
6897 *Asha likes algebra*, we would hope to identify the minimal subtree including just the
6898 concept *like-01* for the word *like*, and the subtree (*p / person :name (n /*
6899 *name :op1 Asha)*) for the word *Asha*.
- 6900 2. **Relation identification** (Figure 13.4b). This involves building a directed graph over
6901 the concepts, where the edges are labeled by the relation type. AMR imposes a
6902 number of constraints on the graph: all concepts must be included, the graph must
6903 be **connected** (there must be a path between every pair of nodes in the undirected
6904 version of the graph), and every node must have at most one outgoing edge of each
6905 type.

6906 Both of these problems are solved by structure prediction. Concept identification re-
6907 quires simultaneously segmenting the text into spans, and labeling each span with a graph
6908 fragment containing one or more concepts. This is done by computing a set of features
6909 for each candidate span *s* and concept labeling *c*, and then returning the labeling with the
6910 highest overall score.

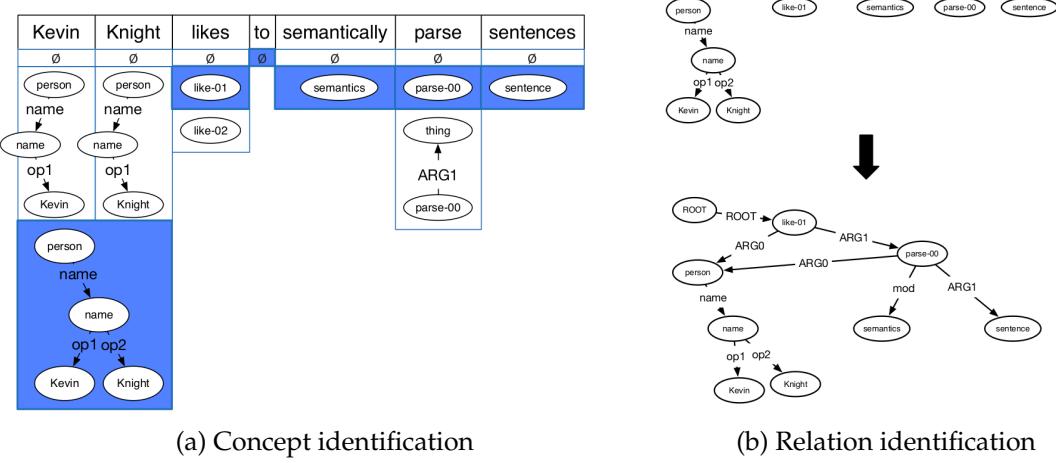


Figure 13.4: Subtasks for Abstract Meaning Representation parsing, from Schneider et al. (2015). [todo: permission]

6911 Relation identification can be formulated as search for the maximum spanning sub-
 6912 graph, under a set of constraints. Each labeled edge has a score, which is computed
 6913 from features of the concepts. We then search for the set of labeled edges that maximizes
 6914 the sum of these scores, under the constraint that the resulting graph is a well-formed
 6915 AMR (Flanigan et al., 2014). This constrained search can be performed by optimization
 6916 techniques such as integer linear programming, as described in § 13.2.2.

6917 **Transition-based parsing** In many cases, AMR parses are structurally similar to syn-
 6918 tactic dependency parses. Figure 13.5 shows one such example. This motivates an alter-
 6919 native approach to AMR parsing: modify the syntactic dependency parse until it looks
 6920 like a good AMR parse. Wang et al. (2015) propose a transition-based method, based on
 6921 incremental modifications to the syntactic dependency tree (transition-based dependency
 6922 parsing is discussed in § 11.3). At each step, the parser performs an action: for example,
 6923 adding an AMR relation label to the current dependency edge, swapping the direction of
 6924 a syntactic dependency edge, or cutting an edge and reattaching the orphaned subtree to
 6925 a new parent. The overall system is trained as a classifier, learning to choose the action as
 6926 would be given by an **oracle** that is capable of reproducing the ground-truth parse.

6927 13.4 Applications of Predicate-Argument Semantics

6928 **Question answering** Factoid questions have answers that are single words or phrases,
 6929 such as *who discovered prions?*, *where was Barack Obama born?*, and *in what year did the Knicks*
 6930 *last win the championship?* Semantic role labeling can be used to answer such questions,

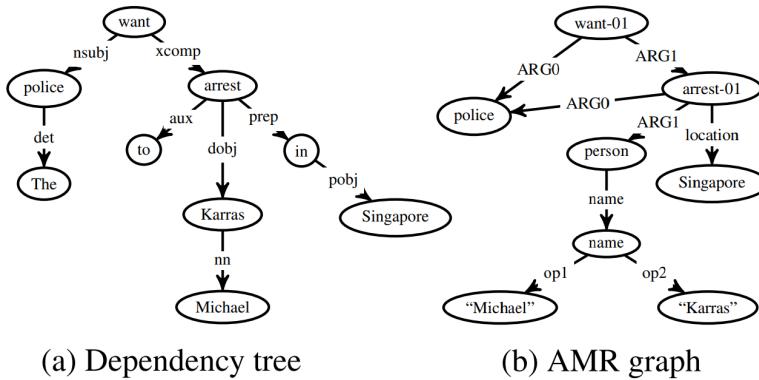


Figure 13.5: Syntactic dependency parse and AMR graph for the sentence *The police want to arrest Michael Karras in Singapore* (borrowed from Wang et al. (2015)) [todo: permission]

6931 by linking questions to sentences in a corpus of text. Shen and Lapata (2007) perform
 6932 FrameNet semantic role labeling on the query, and then construct a weighted **bipartite**
 6933 **graph**¹⁵ between FrameNet semantic roles and the words and phrases in the sentence.
 6934 This is done by first scoring all pairs of semantic roles and assignments, as shown in the
 6935 top half of Figure 13.6. They then find the bipartite edge cover, which is the minimum
 6936 weighted subset of edges such that each vertex has at least one edge, as shown in the
 6937 bottom half of Figure 13.6. After analyzing the question in this manner, Shen and Lapata
 6938 then find semantically-compatible sentences in the corpus, by performing graph matching
 6939 on the bipartite graphs for the question and candidate answer sentences. Finally, the
 6940 *expected answer phrase* in the question — typically the *wh*-word — is linked to a phrase in
 6941 the candidate answer source, and that phrase is returned as the answer.

6942 **Relation extraction** The task of **relation extraction** involves identifying pairs of entities
 6943 for which a given semantic relation holds (see § 17.2. For example, we might like to find
 6944 all pairs (i, j) such that i is the INVENTOR-OF j . PropBank semantic role labeling can
 6945 be applied to this task by identifying sentences whose verb signals the desired relation,
 6946 and then extracting ARG1 and ARG2 as arguments. (To fully solve this task, these argu-
 6947 ments must then be linked to entities, as described in chapter 17.) Christensen et al. (2010)
 6948 compare a semantic role labeling system against a simpler approach based on surface pat-
 6949 terns (Banko et al., 2007). They find that the SRL system is considerably more accurate,
 6950 but that it is several orders of magnitude slower. Conversely, Barnickel et al. (2009) apply
 6951 SENNA, a convolutional neural network SRL system (Collobert and Weston, 2008) to the
 6952 task of identifying biomedical relations (e.g., which genes inhibit or activate each other).

¹⁵A bipartite graph is one in which the vertices can be divided into two disjoint sets, and every edge connects a vertex in one set to a vertex in the other.

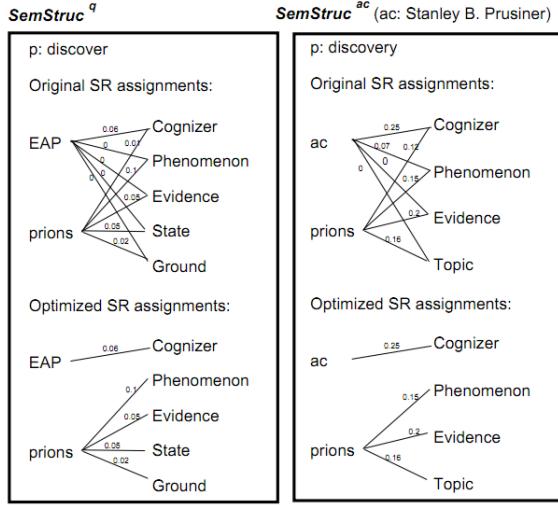


Figure 13.6: FrameNet semantic role labeling is used in factoid question answering, by aligning the semantic roles in the question (q) against those of sentences containing answer candidates (ac). “EAP” is the expected answer phrase, replacing the word *who* in the question. Figure reprinted from Shen and Lapata (2007) [todo: permission]

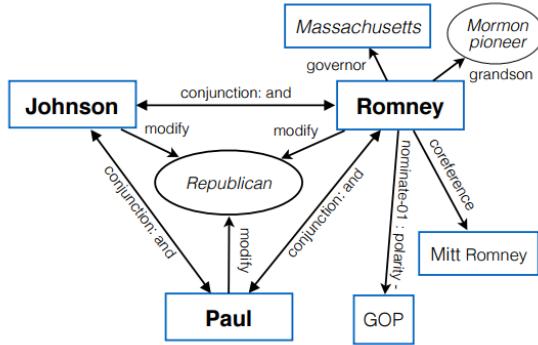


Figure 13.7: Fragment of AMR knowledge network for entity linking. Figure reprinted from Pan et al. (2015) [todo: permission]

6953 In comparison with a strong baseline that applies a set of rules to syntactic dependency
 6954 structures (Fundel et al., 2007), the SRL system is faster but less accurate. One possible
 6955 explanation for these divergent results is that Fundel et al. compare against a baseline
 6956 which is carefully tuned for performance in a relatively narrow domain, while the system
 6957 of Banko et al. is designed to analyze text across the entire web.

6958 **Entity linking** Another core task in information extraction is to link mentions of entities
6959 (e.g., *Republican candidates like Romney, Paul, and Johnson* ...) to entities in a knowledge
6960 base (e.g., LYNDON JOHNSON or GARY JOHNSON). This task, which is described in § 17.1,
6961 is often performed by examining nearby “collaborator” mentions — in this case, *Romney*
6962 and *Paul*. By jointly linking all such mentions, it is possible to arrive at a good overall
6963 solution. Pan et al. (2015) apply AMR to this problem. For each entity, they construct a
6964 knowledge network based on its semantic relations with other mentions within the same
6965 sentence. They then rerank a set of candidate entities, based on the overlap between
6966 the entity’s knowledge network and the semantic relations present in the sentence (Figure
6967 13.7).

6968 **Exercises**

- 6969 1. Write out an event semantic representation for the following sentences. You may
6970 make up your own predicates.
 - 6971 (13.26) *Abigail shares with Max.*
 - 6972 (13.27) *Abigail reluctantly shares a toy with Max.*
 - 6973 (13.28) *Abigail hates to share with Max.*
 - 6974 2. Find the PropBank framesets for *share* and *hate* at <http://verbs.colorado.edu/propbank/framesets-english-aliases/>, and rewrite your answers from the
6975 previous question, using the thematic roles ARG0, ARG1, and ARG2.
 - 6977 3. Compute the syntactic path features for Abigail and Max in each of the example sentences (13.26) and (13.28) in Question 1, with respect to the verb *share*. If you’re not
6978 sure about the parse, you can try an online parser such as <http://nlp.stanford.edu:8080/parser/>.
 - 6981 4. Compute the dependency path features for Abigail and Max in each of the example sentences (13.26) and (13.28) in Question 1, with respect to the verb *share*. Again, if
6982 you’re not sure about the parse, you can try an online parser such as <http://nlp.stanford.edu:8080/parser/>. As a hint, the dependency relation between *share*
6983 and *Max* is OBL according to the Universal Dependency treebank (version 2).
 - 6986 5. PropBank semantic role labeling includes **reference arguments**, such as,
- 6987 (13.29) [AM-LOC The bed] on [R-AM-LOC which] I slept broke.¹⁶

¹⁶Example from 2013 NAACL tutorial slides by Shumin Wu

6988 The label R-AM-LOC indicates that word *which* is a reference to *The bed*, which ex-
 6989 presses the location of the event. Reference arguments must have referents: the tag
 6990 R-AM-LOC can appear only when AM-LOC also appears in the sentence. Show how
 6991 to express this as a linear constraint, specifically for the tag R-AM-LOC. Be sure to
 6992 correctly handle the case in which neither AM-LOC nor R-AM-LOC appear in the
 6993 sentence.

- 6994 6. Explain how to express the constraints on semantic role labeling in Equation 13.8
 6995 and Equation 13.9 in the general form $Az \geq b$.
- 6996 7. Download the FrameNet sample data (<https://framenet.icsi.berkeley.edu/fndrupal/fulltextIndex>), and train a bag-of-words classifier to predict the
 6997 frame that is evoked by each verb in each example. Your classifier should build
 6998 a bag-of-words from the sentence in which the frame-evoking lexical unit appears.
 6999 [*todo: Somehow limit to one or a few lexical units.*] [*todo: use NLTK if possible*]
- 7000 8. Download the PropBank sample data, using NLTK (<http://www.nltk.org/howto/propbank.html>). Use a deep learning toolkit such as PyTorch or DyNet to train an
 7001 LSTM to predict tags. You will have to convert the downloaded instances to a BIO
 7002 sequence labeling representation first.
- 7003 9. Produce the AMR annotations for the following examples:

- 7004 (13.30) The girl likes the boy.
 7005 (13.31) The girl was liked by the boy.
 7006 (13.32) Abigail likes Maxwell Aristotle.
 7007 (13.33) The spy likes the attractive boy.
 7008 (13.34) The girl doesn't like the boy.
 7009 (13.35) The girl likes her dog.

7010 For (13.32), recall that multi-token names are created using op1, op2, etc. You will
 7011 need to consult Banerjee et al. (2013) for (13.34), and Schneider et al. (2015) for
 7012 (13.35). You may assume that *her* refers to *the girl* in this example.

- 7013 10. Using an off-the-shelf PropBank SRL system,¹⁷ build a simplified question answer-
 7014 ing system in the style of Shen and Lapata (2007). Specifically, your system should
 7015 do the following:

¹⁷At the time of writing, the following systems are available: SENNA (<http://ronan.collobert.com/senna/>), Illinois Semantic Role Labeler (https://cogcomp.cs.illinois.edu/page/software_view/SRL), and mate-tools (<https://code.google.com/archive/p/mate-tools/>).

- For each document in a collection, it should apply the semantic role labeler, and should store the output as a tuple.
- For a question, your system should again apply the semantic role labeler. If any of the roles are filled by a *wh*-pronoun, you should mark that role as the expected answer phrase (EAP).
- To answer the question, search for a stored tuple which matches the question as well as possible (same predicate, no incompatible semantic roles, and as many matching roles as possible). Align the EAP against its role filler in the stored tuple, and return this as the answer.

To evaluate your system, download a set of three news articles on the same topic, and write down five factoid questions that should be answerable from the articles. See if your system can answer these questions correctly. (If this problem is assigned to an entire class, you can build a large-scale test set and compare various approaches.)

7032 Chapter 14

7033 Distributional and distributed 7034 semantics

7035 A recurring theme in natural language processing is the complexity of the mapping from
7036 words to meaning. In chapter 4, we saw that a single word form, like *bank*, can have mul-
7037 tiple meanings; conversely, a single meaning may be created by multiple surface forms,
7038 a lexical semantic relationship known as **synonymy**. Despite this complex mapping be-
7039 tween words and meaning, natural language processing systems usually rely on words
7040 as the basic unit of analysis. This is especially true in semantics: the logical and frame
7041 semantic methods from the previous two chapters rely on hand-crafted lexicons that map
7042 from words to semantic predicates. But how can we analyze texts that contain words
7043 that we haven't seen before? This chapter describes methods that learn representations
7044 of word meaning by analyzing unlabeled data, vastly improving the generalizability of
7045 natural language processing systems. The theory that makes it possible to acquire mean-
7046 ingful representations from unlabeled data is the **distributional hypothesis**.

7047 14.1 The distributional hypothesis

7048 Here's a word you may not know: *tezgüino* (the example is from Lin, 1998). If you do not
7049 know the meaning of *tezgüino*, then you are in the same situation as a natural language
7050 processing system when it encounters a word that did not appear in its training data.
7051 Now suppose you see that *tezgüino* is used in the following contexts:

- 7052 (14.1) A bottle of _____ is on the table.
- 7053 (14.2) Everybody likes _____.
- 7054 (14.3) Don't have _____ before you drive.
- 7055 (14.4) We make _____ out of corn.

	(14.1)	(14.2)	(14.3)	(14.4)	...
<i>tezgüino</i>	1	1	1	1	
<i>loud</i>	0	0	0	0	
<i>motor oil</i>	1	0	0	1	
<i>tortillas</i>	0	1	0	1	
<i>choices</i>	0	1	0	0	
<i>wine</i>	1	1	1	0	

Table 14.1: Distributional statistics for *tezgüino* and five related terms

What other words fit into these contexts? How about: *loud*, *motor oil*, *tortillas*, *choices*, *wine*? Each row of Table 14.1 is a vector that summarizes the contextual properties for each word, with a value of one for contexts in which the word can appear, and a value of zero for contexts in which it cannot. Based on these vectors, we can conclude: *wine* is very similar to *tezgüino*; *motor oil* and *tortillas* are fairly similar to *tezgüino*; *loud* is completely different.

These vectors, which we will call **word representations**, describe the **distributional** properties of each word. Does vector similarity imply semantic similarity? This is the **distributional hypothesis**, stated by Firth (1957) as: “You shall know a word by the company it keeps.” The distributional hypothesis has stood the test of time: distributional statistics are a core part of language technology today, because they make it possible to leverage large amounts of unlabeled data to learn about rare words that do not appear in labeled training data.

Distributional statistics have a striking ability to capture lexical semantic relationships such as analogies. Figure 14.1 shows two examples, based on two-dimensional projections of distributional **word embeddings**, discussed later in this chapter. In each case, word-pair relationships correspond to regular linear patterns in this two dimensional space. No labeled data about the nature of these relationships was required to identify this underlying structure.

Distributional semantics are computed from context statistics. **Distributed** semantics are a related but distinct idea: that meaning can be represented by numerical vectors rather than symbolic structures. Distributed representations are often estimated from distributional statistics, as in latent semantic analysis and WORD2VEC, described later in this chapter. However, distributed representations can also be learned in a supervised fashion from labeled data, as in the neural classification models encountered in chapter 3.

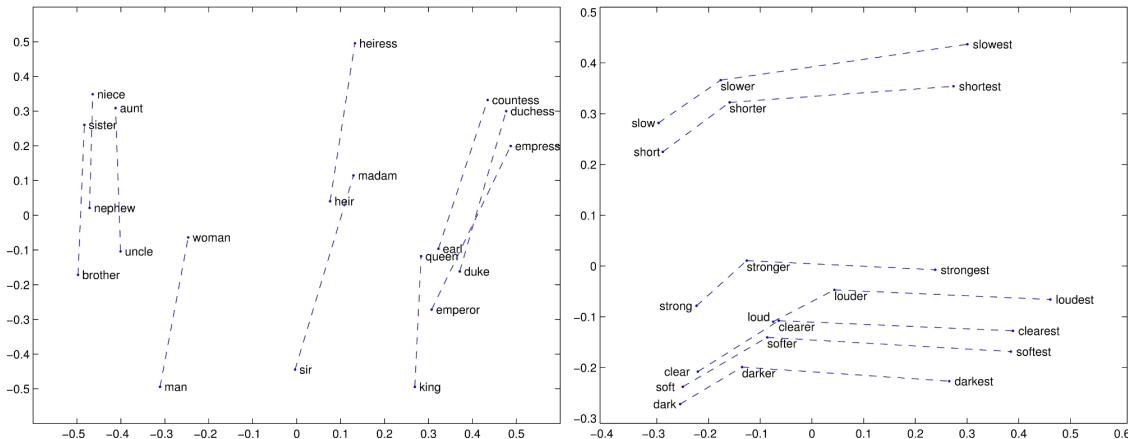


Figure 14.1: Lexical semantic relationships have regular linear structures in two dimensional projections of distributional statistics. From [http://nlp.stanford.edu/projects/glove/.\[todo: redo to make words bigger?\]](http://nlp.stanford.edu/projects/glove/.[todo: redo to make words bigger?])

7081 14.2 Design decisions for word representations

7082 There are many approaches for computing word representations, but most can be distin-
 7083 guished on three main dimensions: the nature of the representation, the source of context-
 7084 ual information, and the estimation procedure.

7085 14.2.1 Representation

7086 Today, the dominant word representations are k -dimensional vectors of real numbers,
 7087 known as **word embeddings**. (The name is due to the fact that each discrete word is em-
 7088 bedded in a continuous vector space.) This representation dates back at least to the late
 7089 1980s (Deerwester et al., 1990), and is used in popular techniques such as WORD2VEC (Mikolov
 7090 et al., 2013).

7091 Word embeddings are well suited for neural networks, where they can be plugged
 7092 in as inputs. They can also be applied in linear classifiers and structure prediction mod-
 7093 els (Turian et al., 2010), although it can be difficult to learn linear models that employ
 7094 real-valued features (Kummerfeld et al., 2015). A popular alternative is bit-string rep-
 7095 resentations, such as **Brown clusters** (§ 14.4), in which each word is represented by a
 7096 variable-length sequence of zeros and ones (Brown et al., 1992).

7097 Another representational question is whether to estimate one embedding per surface
 7098 form (e.g., *bank*), or to estimate distinct embeddings for each word sense or synset. In-
 7099 tuitively, if word representations are to capture the meaning of individual words, then
 7100 words with multiple meanings should have multiple embeddings. This can be achieved

The moment one learns English, complications set in (Alfau, 1999)

Brown Clusters (Brown et al., 1992)	{one}
WORD2VEC (Mikolov et al., 2013) ($h = 2$)	{moment, one, English, complications}
Structured WORD2VEC (Ling et al., 2015) ($h = 2$)	$\{(moment, -2), (one, -1), (English, +1), (complications, +2)\}$
Dependency contexts (Levy and Goldberg, 2014)	$\{(one, \text{NSUBJ}), (English, \text{DOBJ}), (moment, \text{ACL}^{-1})\}$

Table 14.2: Contexts for the word *learns*, according to various word representations. For dependency context, *(one, NSUBJ)* means that there is a relation of type NSUBJ (nominal subject) **to** the word *one*, and *(moment, ACL⁻¹)* means that there is a relation of type ACL (adjectival clause) **from** the word *moment*.

7101 by integrating unsupervised clustering with word embedding estimation (Huang and
 7102 Yates, 2012; Li and Jurafsky, 2015). However, Arora et al. (2016) argue that it is unnec-
 7103 essary to model distinct word senses explicitly, because the embeddings for each surface
 7104 form are a linear combination of the embeddings of the underlying senses.

7105 14.2.2 Context

7106 The distributional hypothesis says that word meaning is related to the “contexts” in which
 7107 the word appears, but context can be defined in many ways. In the *tezgiiino* example, con-
 7108 texts are entire sentences, but in practice there are far too many sentences. At the oppo-
 7109 site extreme, the context could be defined as the immediately preceding word; this is the
 7110 context considered in Brown clusters. WORD2VEC takes an intermediate approach, using
 7111 local neighborhoods of words (e.g., $h = 5$) as contexts (Mikolov et al., 2013). Contexts
 7112 can also be much larger: for example, in **latent semantic analysis**, each word’s context
 7113 vector includes an entry per document, with a value of one if the word appears in the
 7114 document (Deerwester et al., 1990); in **explicit semantic analysis**, these documents are
 7115 Wikipedia pages (Gabrilovich and Markovitch, 2007).

7116 Words in context can be labeled by their position with respect to the target word w_m
 7117 (e.g., two words before, one word after), which makes the resulting word representations
 7118 more sensitive to syntactic differences (Ling et al., 2015). Another way to incorporate
 7119 syntax is to perform parsing as a preprocessing step, and then form context vectors from
 7120 the dependency edges (Levy and Goldberg, 2014) or predicate-argument relations (Lin,
 7121 1998). The resulting context vectors for several of these methods are shown in Table 14.2.

7122 The choice of context has a profound effect on the resulting representations, which

7123 can be viewed in terms of word similarity. Applying latent semantic analysis (§ 14.3) to
 7124 contexts of size $h = 2$ and $h = 30$ yields the following nearest-neighbors for the word
 7125 *dog*:¹

- 7126 • ($h = 2$): *cat, horse, fox, pet, rabbit, pig, animal, mongrel, sheep, pigeon*
- 7127 • ($h = 30$): *kennel, puppy, pet, bitch, terrier, rottweiler, canine, cat, to bark, Alsatian*

7128 Which word list is better? Each word in the $h = 2$ list is an animal, reflecting the fact that
 7129 locally, the word *dog* tends to appear in the same contexts as other animal types (e.g., *pet*
 7130 *the dog, feed the dog*). In the $h = 30$ list, nearly everything is dog-related, including specific
 7131 breeds such as *rottweiler* and *Alsatian*. The list also includes words that are not animals
 7132 (*kennel*), and in one case (*to bark*), is not a noun at all. The 2-word context window is more
 7133 sensitive to syntax, while the 30-word window is more sensitive to topic.

7134 14.2.3 Estimation

7135 Word embeddings are estimated by optimizing some objective: the likelihood of a set of
 7136 unlabeled data (or a closely related quantity), or the reconstruction of a matrix of context
 7137 counts, similar to Table 14.1.

7138 **Maximum likelihood estimation** Likelihood-based optimization is derived from the
 7139 objective $\log p(\mathbf{w}; \mathbf{U})$, where $\mathbf{U} \in \mathbb{R}^{K \times V}$ is matrix of word embeddings, and $\mathbf{w} =$
 7140 $\{\mathbf{w}_m\}_{m=1}^M$ is a corpus, represented as a list of M tokens. Recurrent neural network lan-
 7141 guage models (§ 6.3) optimize this objective directly, backpropagating to the input word
 7142 embeddings through the recurrent structure. However, state-of-the-art word embeddings
 7143 employ huge corpora with hundreds of billions of tokens, and recurrent architectures are
 7144 difficult to scale to such data. As a result, likelihood-based word embeddings are usually
 7145 based on simplified likelihoods or heuristic approximations.

Matrix factorization The matrix $\mathbf{C} = \{\text{count}(i, j)\}$ stores the co-occurrence counts of
 word i and context j . Word representations can be obtained by approximately factoring
 this matrix, so that $\text{count}(i, j)$ is approximated by a function of a word embedding \mathbf{u}_i and
 a context embedding \mathbf{v}_j . These embeddings can be obtained by minimizing the norm of
 the reconstruction error,

$$\min_{\mathbf{u}, \mathbf{v}} \|\mathbf{C} - \tilde{\mathbf{C}}(\mathbf{u}, \mathbf{v})\|_F, \quad [14.1]$$

¹The example is from lecture slides by Marco Baroni, Alessandro Lenci, and Stefan Evert, who applied latent semantic analysis to the British National Corpus. You can find an online demo here: <http://clic.cimec.unitn.it/infomap-query/>

7146 where $\tilde{\mathbf{C}}(\mathbf{u}, \mathbf{v})$ is the approximate reconstruction resulting from the embeddings \mathbf{u} and
 7147 \mathbf{v} , and $\|\mathbf{X}\|_F$ indicates the Frobenius norm, $\sum_{i,j} x_{i,j}^2$. Rather than factoring the matrix of
 7148 word-context counts directly, it is often helpful to transform these counts using information-
 7149 theoretic metrics such as **pointwise mutual information** (PMI), described in the next sec-
 7150 tion.

7151 14.3 Latent semantic analysis

Latent semantic analysis (LSA) is one of the oldest approaches to distributed semantics (Deerwester et al., 1990). It induces continuous vector representations of words by factoring a matrix of word and context counts, using **truncated singular value decomposition** (SVD),

$$\min_{\mathbf{U} \in \mathbb{R}^{V \times K}, \mathbf{S} \in \mathbb{R}^{K \times K}, \mathbf{V} \in \mathbb{R}^{|\mathcal{C}| \times K}} \|\mathbf{C} - \mathbf{USV}^\top\|_F \quad [14.2]$$

$$\text{s.t. } \mathbf{U}^\top \mathbf{U} = \mathbb{I} \quad [14.3]$$

$$\mathbf{V}^\top \mathbf{V} = \mathbb{I} \quad [14.4]$$

$$\forall i \neq j, \mathbf{S}_{i,j} = 0, \quad [14.5]$$

7152 where V is the size of the vocabulary, $|\mathcal{C}|$ is the number of contexts, and K is size of the
 7153 resulting embeddings, which are set equal to the rows of the matrix \mathbf{U} . The matrix \mathbf{S} is
 7154 constrained to be diagonal (these diagonal elements are called the singular values), and
 7155 the columns of the product \mathbf{SV}^\top provide descriptions of the contexts. Each element $c_{i,j}$ is
 7156 then reconstructed as a **bilinear product**,

$$c_{i,j} \approx \sum_{k=1}^K u_{i,k} s_k v_{j,k}. \quad [14.6]$$

7157 The objective is to minimize the sum of squared approximation errors. The orthonormality
 7158 constraints $\mathbf{U}^\top \mathbf{U} = \mathbf{V}^\top \mathbf{V} = \mathbb{I}$ ensure that all pairs of dimensions in \mathbf{U} and \mathbf{V} are
 7159 uncorrelated, so that each dimension conveys unique information. Efficient implemen-
 7160 tations of truncated singular value decomposition are available in numerical computing
 7161 packages such as `scipy` and `matlab`.²

Latent semantic analysis is most effective when the count matrix is transformed before the application of SVD. One such transformation is **pointwise mutual information** (PMI; Church and Hanks, 1990), which captures the degree of association between word i and

²An important implementation detail is to represent \mathbf{C} as a **sparse matrix**, so that the storage cost is equal to the number of non-zero entries, rather than the size $V \times |\mathcal{C}|$.

context j ,

$$\text{PMI}(i, j) = \log \frac{p(i, j)}{p(i)p(j)} = \log \frac{p(i | j)p(j)}{p(i)p(j)} = \log \frac{p(i | j)}{p(i)} \quad [14.7]$$

$$= \log \text{count}(i, j) - \log \sum_{i'=1}^V \text{count}(i', j) \quad [14.8]$$

$$- \log \sum_{j' \in \mathcal{C}} \text{count}(i, j') + \log \sum_{i'=1}^V \sum_{j' \in \mathcal{C}} \text{count}(i', j'). \quad [14.9]$$

The pointwise mutual information can be viewed as the logarithm of the ratio of the conditional probability of word i in context j to the marginal probability of word i in all contexts. When word i is statistically associated with context j , the ratio will be greater than one, so $\text{PMI}(i, j) > 0$. The PMI transformation focuses latent semantic analysis on reconstructing strong word-context associations, rather than on reconstructing large counts.

The PMI is negative when a word and context occur together less often than if they were independent, but such negative correlations are unreliable because counts of rare events have high variance. Furthermore, the PMI is undefined when $\text{count}(i, j) = 0$. One solution to these problems is to use the **Positive PMI** (PPMI),

$$\text{PPMI}(i, j) = \begin{cases} \text{PMI}(i, j), & p(i | j) > p(i) \\ 0, & \text{otherwise.} \end{cases} \quad [14.10]$$

Bullinaria and Levy (2007) compare a range of matrix transformations for latent semantic analysis, using a battery of tasks related to word meaning and word similarity (for more on evaluation, see § 14.6). They find that PPMI-based latent semantic analysis yields strong performance on a battery of tasks related to word meaning: for example, PPMI-based LSA vectors can be used to solve multiple-choice word similarity questions from the Test of English as a Foreign Language (TOEFL), obtaining 85% accuracy.

14.4 Brown clusters

Learning algorithms like perceptron and conditional random fields often perform better with discrete feature vectors. A simple way to obtain discrete representations from distributional statistics is by clustering (§ 5.1.1), so that words in the same cluster have similar distributional statistics. This can help in downstream tasks, by sharing features between all words in the same cluster. However, there is an obvious tradeoff: if the number of clusters is too small, the words in each cluster will not have much in common; if the number of clusters is too large, then the learner will not see enough examples from each cluster to generalize.

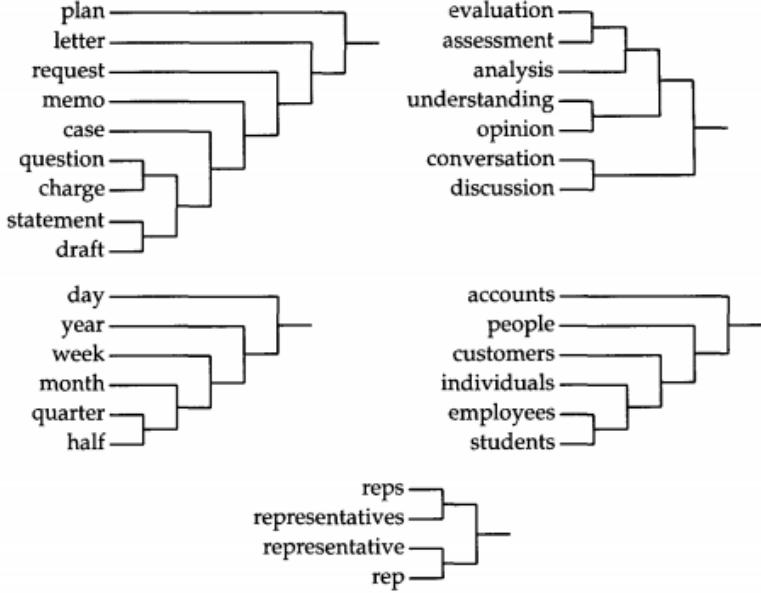


Figure 14.2: Some subtrees produced by bottom-up Brown clustering (Miller et al., 2004) on news text [todo: permission]

7186 A solution to this problem is **hierarchical clustering**: using the distributional statistics
 7187 to induce a tree-structured representation. Fragments of **Brown cluster** trees are shown in
 7188 Figure 14.2 and Table 14.3. Each word’s representation consists of a binary string describ-
 7189 ing a path through the tree: 0 for taking the left branch, and 1 for taking the right branch.
 7190 In the subtree in the upper right of the figure, the representation of the word *conversation*
 7191 is 10; the representation of the word *assessment* is 0001. Bitstring prefixes capture simila-
 7192 rity at varying levels of specificity, and it is common to use the first eight, twelve, sixteen,
 7193 and twenty bits as features in tasks such as named entity recognition (Miller et al., 2004)
 7194 and dependency parsing (Koo et al., 2008).

Hierarchical trees can be induced from a likelihood-based objective, using a discrete latent variable $k_i \in \{1, 2, \dots, K\}$ to represent the cluster of word i :

$$\log p(\mathbf{w}; \mathbf{k}) \approx \sum_{m=1}^M \log p(w_m | w_{m-1}; \mathbf{k}) \quad [14.11]$$

$$\triangleq \sum_{m=1}^M \log p(w_m | k_{w_m}) + \log p(k_{w_m} | k_{w_{m-1}}). \quad [14.12]$$

7195 This is similar to a hidden Markov model, with the crucial difference that each word can

bitstring	ten most frequent words
01111010 0111	<i>excited thankful grateful stoked pumped anxious hyped psyched exited geeked</i>
01111010 100	<i>talking talkin complaining talkn bitching tlkn tlkin bragging rav- ing +k</i>
01111010 1010	<i>thinking thinkin dreaming worrying thinkn speakin reminiscing dreamin daydreaming fantasizing</i>
01111010 1011	<i>saying sayin suggesting stating sayn jokin talmbout implying insisting 5'2</i>
01111010 1100	<i>wonder dunno wondered duno donno dno doно wonda wounder dunnoe</i>
01111010 1101	<i>wondering wonders debating deciding pondering unsure won- derin debatin woundering wondern</i>
01111010 1110	<i>sure suree suuure suure sure- surre sures shuree</i>

Table 14.3: Fragment of a Brown clustering of Twitter data (Owoputi et al., 2013). Each row is a leaf in the tree, showing the ten most frequent words. This part of the tree emphasizes verbs of communicating and knowing, especially in the present participle. Each leaf node includes orthographic variants (*thinking*, *thinkin*, *thinkn*), semantically related terms (*excited*, *thankful*, *grateful*), and some outliers (*5'2*, *+k*). See http://www.cs.cmu.edu/~ark/TweetNLP/cluster_viewer.html for more.

7196 be emitted from only a single cluster: $\forall k \neq k_{w_m}, p(w_m | k) = 0$.

Using the objective in Equation 14.12, the Brown clustering tree can be constructed from the bottom up: begin with each word in its own cluster, and incrementally merge clusters until only a single cluster remains. At each step, we merge the pair of clusters such that the objective in Equation 14.12 is maximized. Although the objective seems to involve a sum over the entire corpus, the score for each merger can be computed from the cluster-to-cluster co-occurrence counts. These counts can be updated incrementally as the clustering proceeds. The optimal merge at each step can be shown to maximize the **average mutual information**,

$$I(\mathbf{k}) = \sum_{k_1=1}^K \sum_{k_2=1}^K p(k_1, k_2) \times \text{PMI}(k_1, k_2) \quad [14.13]$$

$$p(k_1, k_2) = \frac{\text{count}(k_1, k_2)}{\sum_{k_{1'}=1}^K \sum_{k_{2'}=1}^K \text{count}(k_{1'}, k_{2'})},$$

7197 where $p(k_1, k_2)$ is the joint probability of a bigram involving a word in cluster k_1 followed
7198 by a word in k_2 . This probability and the PMI are both computed from the co-occurrence

Algorithm 17 Exchange clustering algorithm. Assumes that words are sorted by frequency, and that MAXMI finds the cluster pair whose merger maximizes the mutual information, as defined in Equation 14.13.

```

procedure EXCHANGECLUSTERING({count( $\cdot, \cdot$ )},  $K$ )
    for  $i \in 1 \dots K$  do                                 $\triangleright$  Initialization
         $k_i \leftarrow i$ ,  $i = 1, 2, \dots, K$ 
        for  $j \in 1 \dots K$  do
             $c_{i,j} \leftarrow \text{count}(i, j)$ 
         $\tau \leftarrow \{(i)\}_{i=1}^K$ 
        for  $i \in \{K + 1, K + 2, \dots, V\}$  do       $\triangleright$  Iteratively add each word to the clustering
             $\tau \leftarrow \tau \cup (i)$ 
            for  $k \in \tau$  do
                 $c_{k,i} \leftarrow \text{count}(k, i)$ 
                 $c_{i,k} \leftarrow \text{count}(i, k)$ 
             $\hat{i}, \hat{j} \leftarrow \text{MAXMI}(\mathbf{C})$ 
             $\tau, \mathbf{C} \leftarrow \text{MERGE}(\hat{i}, \hat{j}, \mathbf{C}, \tau)$ 
        repeat                                          $\triangleright$  Merge the remaining clusters into a tree
             $\hat{i}, \hat{j} \leftarrow \text{MAXMI}(\mathbf{C}, \tau)$ 
             $\tau, \mathbf{C} \leftarrow \text{MERGE}(\hat{i}, \hat{j}, \mathbf{C}, \tau)$ 
        until  $|\tau| = 1$ 
        return  $\tau$ 
procedure MERGE( $i, j, \mathbf{C}, \tau$ )
     $\tau \leftarrow \tau \setminus i \setminus j \cup (i, j)$            $\triangleright$  Merge the clusters in the tree
    for  $k \in \tau$  do                                 $\triangleright$  Aggregate the counts across the merged clusters
         $c_{k,(i,j)} \leftarrow c_{k,i} + c_{k,j}$ 
         $c_{(i,j),k} \leftarrow c_{i,k} + c_{j,k}$ 
    return  $\tau, \mathbf{C}$ 

```

7199 counts between clusters. After each merger, the co-occurrence vectors for the merged
 7200 clusters are simply added up, so that the next optimal merger can be found efficiently.

7201 This bottom-up procedure requires iterating over the entire vocabulary, and evaluating
 7202 K_t^2 possible mergers at each step, where K_t is the current number of clusters at step t
 7203 of the algorithm. Furthermore, computing the score for each merger involves a sum over
 7204 K_t^2 clusters. The maximum number of clusters is $K_0 = V$, which occurs when every word
 7205 is in its own cluster at the beginning of the algorithm. The time complexity is thus $\mathcal{O}(V^5)$.

7206 To avoid this complexity, practical implementations use a heuristic approximation
 7207 called **exchange clustering**. The K most common words are placed in clusters of their
 7208 own at the beginning of the process. We then consider the next most common word, and

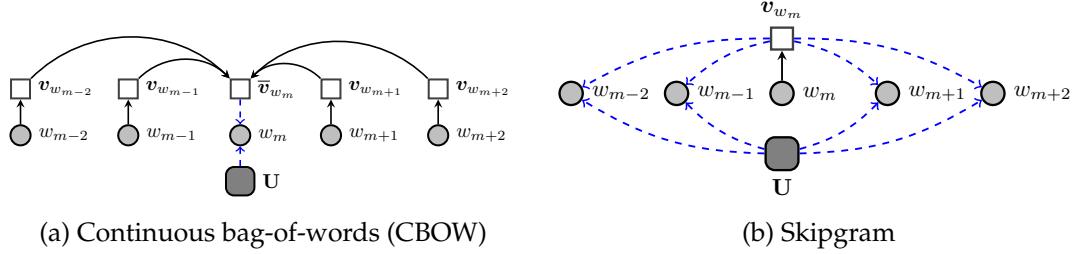


Figure 14.3: The CBOW and skipgram variants of WORD2VEC. The parameter \mathbf{U} is the matrix of word embeddings, and each \mathbf{v}_m is the context embedding for word w_m .

7209 merge it with one of the existing clusters. This continues until the entire vocabulary has
 7210 been incorporated, at which point the K clusters are merged down to a single cluster,
 7211 forming a tree. The algorithm never considers more than $K + 1$ clusters at any step, and
 7212 the complexity is $\mathcal{O}(VK + V \log V)$, with the second term representing the cost of sorting
 7213 the words at the beginning of the algorithm.

7214 14.5 Neural word embeddings

7215 Neural word embeddings combine aspects of the previous two methods: like latent se-
 7216 mantic analysis, they are a continuous vector representation; like Brown clusters, they are
 7217 trained from a likelihood-based objective. Let the vector \mathbf{u}_i represent the K -dimensional
 7218 **embedding** for word i , and let \mathbf{v}_j represent the K -dimensional embedding for context
 7219 j . The inner product $\mathbf{u}_i \cdot \mathbf{v}_j$ represents the compatibility between word i and context j .
 7220 By incorporating this inner product into an approximation to the log-likelihood of a cor-
 7221 pus, it is possible to estimate both parameters by backpropagation. WORD2VEC (Mikolov
 7222 et al., 2013) includes two such approximations: continuous bag-of-words (CBOW) and
 7223 skipgrams.

7224 14.5.1 Continuous bag-of-words (CBOW)

7225 In recurrent neural network language models, each word w_m is conditioned on a recurrently-
 7226 updated state vector, which is based on word representations going all the way back to the
 7227 beginning of the text. The **continuous bag-of-words (CBOW)** model is a simplification:
 7228 the local context is computed as an average of embeddings for words in the immediate
 7229 neighborhood $m - h, m - h + 1, \dots, m + h - 1, m + h$,

$$\bar{\mathbf{v}}_m = \frac{1}{2h} \sum_{n=1}^h \mathbf{v}_{w_{m+n}} + \mathbf{v}_{w_{m-n}}. \quad [14.14]$$

7230 Thus, CBOW is a bag-of-words model, because the order of the context words does not
 7231 matter; it is continuous, because rather than conditioning on the words themselves, we
 7232 condition on a continuous vector constructed from the word embeddings. The parameter
 7233 h determines the neighborhood size, which Mikolov et al. (2013) set to $h = 4$.

The CBOW model optimizes an approximation to the corpus log-likelihood,

$$\log p(\mathbf{w}) \approx \sum_{m=1}^M \log p(w_m | w_{m-h}, w_{m-h+1}, \dots, w_{m+h-1}, w_{m+h}) \quad [14.15]$$

$$= \sum_{m=1}^M \log \frac{\exp(\mathbf{u}_{w_m} \cdot \bar{\mathbf{v}}_m)}{\sum_{j=1}^V \exp(\mathbf{u}_j \cdot \bar{\mathbf{v}}_m)} \quad [14.16]$$

$$= \sum_{m=1}^M \mathbf{u}_{w_m} \cdot \bar{\mathbf{v}}_m - \log \sum_{j=1}^V \exp(\mathbf{u}_j \cdot \bar{\mathbf{v}}_m). \quad [14.17]$$

7234 14.5.2 Skipgrams

In the CBOW model, words are predicted from their context. In the **skipgram** model, the context is predicted from the word, yielding the objective:

$$\log p(\mathbf{w}) \approx \sum_{m=1}^M \sum_{n=1}^{h_m} \log p(w_{m-n} | w_m) + \log p(w_{m+n} | w_m) \quad [14.18]$$

$$= \sum_{m=1}^M \sum_{n=1}^{h_m} \log \frac{\exp(\mathbf{u}_{w_{m-n}} \cdot \mathbf{v}_{w_m})}{\sum_{j=1}^V \exp(\mathbf{u}_j \cdot \mathbf{v}_{w_m})} + \log \frac{\exp(\mathbf{u}_{w_{m+n}} \cdot \mathbf{v}_{w_m})}{\sum_{j=1}^V \exp(\mathbf{u}_j \cdot \mathbf{v}_{w_m})} \quad [14.19]$$

$$= \sum_{m=1}^M \sum_{n=1}^{h_m} \mathbf{u}_{w_{m-n}} \cdot \mathbf{v}_{w_m} + \mathbf{u}_{w_{m+n}} \cdot \mathbf{v}_{w_m} - 2 \log \sum_{j=1}^V \exp(\mathbf{u}_j \cdot \mathbf{v}_{w_m}). \quad [14.20]$$

7235 In the skipgram approximation, each word is generated multiple times; each time it is con-
 7236 ditioned only on a single word. This makes it possible to avoid averaging the word vec-
 7237 tors, as in the CBOW model. The local neighborhood size h_m is randomly sampled from
 7238 a uniform categorical distribution over the range $\{1, 2, \dots, h_{\max}\}$; Mikolov et al. (2013) set
 7239 $h_{\max} = 10$. Because the neighborhood grows outward with h , this approach has the effect
 7240 of weighting near neighbors more than distant ones. Skipgram performs better on most
 7241 evaluations than CBOW (see § 14.6 for details of how to evaluate word representations),
 7242 but CBOW is faster to train (Mikolov et al., 2013).

7243 14.5.3 Computational complexity

7244 The WORD2VEC models can be viewed as an efficient alternative to recurrent neural net-
 7245 work language models, which involve a recurrent state update whose time complexity

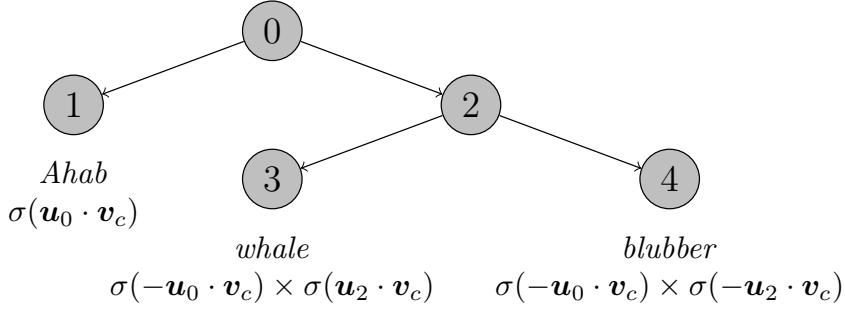


Figure 14.4: A fragment of a hierarchical softmax tree. The probability of each word is computed as a product of probabilities of local branching decisions in the tree.

is quadratic in the size of the recurrent state vector. CBOW and skipgram avoid this computation, and incur only a linear time complexity in the size of the word and context representations. However, all three models compute a normalized probability over word tokens; a naïve implementation of this probability requires summing over the entire vocabulary. The time complexity of this sum is $\mathcal{O}(V \times K)$, which dominates all other computational costs. There are two solutions: **hierarchical softmax**, a tree-based computation that reduces the cost to a logarithm of the size of the vocabulary; and **negative sampling**, an approximation that eliminates the dependence on vocabulary size. Both methods are also applicable to RNN language models.

14.5.3.1 Hierarchical softmax

In Brown clustering, the vocabulary is organized into a binary tree. Mnih and Hinton (2008) show that the normalized probability over words in the vocabulary can be reparametrized as a probability over paths through such a tree. This **hierarchical softmax** probability is computed as a product of binary decisions over whether to move left or right through the tree, with each binary decision represented as a sigmoid function of the inner product between the context embedding \mathbf{v}_c and an output embedding associated with the node \mathbf{u}_n ,

$$\Pr(\text{left at } n \mid c) = \sigma(\mathbf{u}_n \cdot \mathbf{v}_c) \quad [14.21]$$

$$\Pr(\text{right at } n \mid c) = 1 - \sigma(\mathbf{u}_n \cdot \mathbf{v}_c) = \sigma(-\mathbf{u}_n \cdot \mathbf{v}_c), \quad [14.22]$$

where σ refers to the sigmoid function, $\sigma(x) = \frac{1}{1+\exp(-x)}$. The range of the sigmoid is the interval $(0, 1)$, and $1 - \sigma(x) = \sigma(-x)$.

As shown in Figure 14.4, the probability of generating each word is redefined as the product of the probabilities across its path. The sum of all such path probabilities is guaranteed to be one, for any context vector $\mathbf{v}_c \in \mathbb{R}^K$. In a balanced binary tree, the depth is

7261 logarithmic in the number of leaf nodes, and thus the number of multiplications is equal
 7262 to $\mathcal{O}(\log V)$. The number of non-leaf nodes is equal to $\mathcal{O}(2V - 1)$, so the number of pa-
 7263 rameters to be estimated increases by only a small multiple. The tree can be constructed
 7264 using an incremental clustering procedure similar to hierarchical Brown clusters (Mnih
 7265 and Hinton, 2008), or by using the Huffman (1952) encoding algorithm for lossless com-
 7266 pression.

7267 **14.5.3.2 Negative sampling**

Likelihood-based methods are computationally intensive because each probability must be normalized over the vocabulary. These probabilities are based on scores for each word in each context, and it is possible to design an alternative objective that is based on these scores more directly: we seek word embeddings that maximize the score for the word that was really observed in each context, while minimizing the scores for a set of randomly selected **negative samples**:

$$\psi(i, j) = \log \sigma(\mathbf{u}_i \cdot \mathbf{v}_j) + \sum_{i' \in \mathcal{W}_{\text{neg}}} \log(1 - \sigma(\mathbf{u}_{i'} \cdot \mathbf{v}_j)), \quad [14.23]$$

7268 where $\psi(i, j)$ is the score for word i in context j , and \mathcal{W}_{neg} is the set of negative samples.
 7269 The objective is to maximize the sum over the corpus, $\sum_{m=1}^M \psi(w_m, c_m)$, where w_m is
 7270 token m and c_m is the associated context.

7271 The set of negative samples \mathcal{W}_{neg} is obtained by sampling from a unigram language
 7272 model. Mikolov et al. (2013) construct this unigram language model by exponentiating
 7273 the empirical word probabilities, setting $\hat{p}(i) \propto (\text{count}(i))^{\frac{3}{4}}$. This has the effect of redi-
 7274 tributing probability mass from common to rare words. The number of negative samples
 7275 increases the time complexity of training by a constant factor. Mikolov et al. (2013) report
 7276 that 5-20 negative samples works for small training sets, and that two to five samples
 7277 suffice for larger corpora.

7278 **14.5.4 Word embeddings as matrix factorization**

7279 The negative sampling objective in Equation 14.23 can be justified as an efficient approx-
 7280 imation to the log-likelihood, but it is also closely linked to the matrix factorization ob-
 7281 jective employed in latent semantic analysis. For a matrix of word-context pairs in which
 7282 all counts are non-zero, negative sampling is equivalent to factorization of the matrix M ,
 7283 where $M_{ij} = \text{PMI}(i, j) - \log k$: each cell in the matrix is equal to the pointwise mutual
 7284 information of the word and context, shifted by $\log k$, with k equal to the number of neg-
 7285 ative samples (Levy and Goldberg, 2014). For word-context pairs that are not observed in
 7286 the data, the pointwise mutual information is $-\infty$, but this can be addressed by consid-
 7287 ering only PMI values that are greater than $\log k$, resulting in a matrix of **shifted positive**

7288 **pointwise mutual information,**

$$M_{ij} = \max(0, \text{PMI}(i, j) - \log k). \quad [14.24]$$

7289 Word embeddings are obtained by factoring this matrix with truncated singular value
7290 decomposition.

GloVe (“global vectors”) are a closely related approach (Pennington et al., 2014), in which the matrix to be factored is constructed from log co-occurrence counts, $M_{ij} = \log \text{count}(i, j)$. The word embeddings are estimated by minimizing the sum of squares,

$$\begin{aligned} \min_{\mathbf{u}, \mathbf{v}, b, \tilde{b}} \quad & \sum_{j=1}^V \sum_{j \in C} f(M_{ij}) \left(\widehat{\log M_{ij}} - \log M_{ij} \right)^2 \\ \text{s.t.} \quad & \widehat{\log M_{ij}} = \mathbf{u}_i \cdot \mathbf{v}_j + b_i + \tilde{b}_j, \end{aligned} \quad [14.25]$$

7291 where b_i and \tilde{b}_j are offsets for word i and context j , which are estimated jointly with the
7292 embeddings \mathbf{u} and \mathbf{v} . The weighting function $f(M_{ij})$ is set to be zero at $M_{ij} = 0$, thus
7293 avoiding the problem of taking the logarithm of zero counts; it saturates at $M_{ij} = m_{\max}$,
7294 thus avoiding the problem of overcounting common word-context pairs. This heuristic
7295 turns out to be critical to the method’s performance.

7296 The time complexity of sparse matrix reconstruction is determined by the number of
7297 non-zero word-context counts. Pennington et al. (2014) show that this number grows
7298 sublinearly with the size of the dataset: roughly $\mathcal{O}(N^{0.8})$ for typical English corpora. In
7299 contrast, the time complexity of WORD2VEC is linear in the corpus size. Computing the co-
7300 occurrence counts also requires linear time in the size of the corpus, but this operation can
7301 easily be parallelized using MapReduce-style algorithms (Dean and Ghemawat, 2008).

7302 14.6 Evaluating word embeddings

7303 Distributed word representations can be evaluated in two main ways. **Intrinsic** evalua-
7304 tions test whether the representations cohere with our intuitions about word meaning.
7305 **Extrinsic** evaluations test whether they are useful for downstream tasks, such as sequence
7306 labeling.

7307 14.6.1 Intrinsic evaluations

7308 A basic question for word embeddings is whether the similarity of words i and j is re-
7309 flected in the similarity of the vectors \mathbf{u}_i and \mathbf{u}_j . **Cosine similarity** is typically used to
7310 compare two word embeddings,

$$\cos(\mathbf{u}_i, \mathbf{u}_j) = \frac{\mathbf{u}_i \cdot \mathbf{u}_j}{\|\mathbf{u}_i\|_2 \times \|\mathbf{u}_j\|_2}. \quad [14.26]$$

word 1	word 2	similarity
<i>love</i>	<i>sex</i>	6.77
<i>stock</i>	<i>jaguar</i>	0.92
<i>money</i>	<i>cash</i>	9.15
<i>development</i>	<i>issue</i>	3.97
<i>lad</i>	<i>brother</i>	4.46

Table 14.4: Subset of the WS-353 (Finkelstein et al., 2002) dataset of word similarity ratings (examples from Faruqui et al. (2016)).

7311 For any embedding method, we can evaluate whether the cosine similarity of word em-
 7312 beddings is correlated with human judgments of word similarity. The WS-353 dataset (Finkel-
 7313 stein et al., 2002) includes similarity scores for 353 word pairs (Table 14.4). To test the
 7314 accuracy of embeddings for rare and morphologically complex words, Luong et al. (2013)
 7315 introduce a dataset of “rare words.” Outside of English, word similarity resources are
 7316 limited, mainly consisting of translations of WS-353.

7317 Word analogies (e.g., *king:queen :: man:woman*) have also been used to evaluate word
 7318 embeddings (Mikolov et al., 2013). In this evaluation, the system is provided with the first
 7319 three parts of the analogy ($i_1 : j_1 :: i_2 : ?$), and the final element is predicted by finding the
 7320 word embedding most similar to $\mathbf{u}_{i_1} - \mathbf{u}_{j_1} + \mathbf{u}_{i_2}$. Another evaluation tests whether word
 7321 embeddings are related to broad lexical semantic categories called **supersenses** (Ciaramita
 7322 and Johnson, 2003): verbs of motion, nouns that describe animals, nouns that describe
 7323 body parts, and so on. These supersenses are annotated for English synsets in Word-
 7324 Net (Fellbaum, 2010). This evaluation is implemented in the `qvec` metric, which tests
 7325 whether the matrix of supersenses can be reconstructed from the matrix of word embed-
 7326 dings (Tsvetkov et al., 2015).

7327 Levy et al. (2015) compared several dense word representations for English — includ-
 7328 ing latent semantic analysis, WORD2VEC, and GloVe — using six word similarity metrics
 7329 and two analogy tasks. None of the embeddings outperformed the others on every task,
 7330 but skipgrams were the most broadly competitive. Hyperparameter tuning played a key
 7331 role: any method will perform badly if the wrong hyperparameters are used. Relevant
 7332 hyperparameters include the embedding size, as well as algorithm-specific details such
 7333 as the neighborhood size and the number of negative samples.

7334 14.6.2 Extrinsic evaluations

7335 Word representations contribute to downstream tasks like sequence labeling and docu-
 7336 ment classification by enabling generalization across words. The use of distributed repre-
 7337 sentations as features is a form of **semi-supervised learning**, in which performance on a

7338 supervised learning problem is augmented by learning distributed representations from
 7339 unlabeled data (Miller et al., 2004; Koo et al., 2008; Turian et al., 2010). These **pre-trained**
 7340 **word representations** can be used as features in a linear prediction model, or as the input
 7341 layer in a neural network, such as a Bi-LSTM tagging model (§ 7.6). Word representations
 7342 can be evaluated by the performance of the downstream systems that consume them:
 7343 for example, GloVe embeddings are convincingly better than Latent Semantic Analysis
 7344 as features in the downstream task of named entity recognition (Pennington et al., 2014).
 7345 Unfortunately, extrinsic and intrinsic evaluations do not always point in the same direc-
 7346 tion, and the best word representations for one downstream task may perform poorly on
 7347 another task (Schnabel et al., 2015).

7348 When word representations are updated from labeled data in the downstream task,
 7349 they are said to be **fine-tuned**. When labeled data is plentiful, pre-training may be un-
 7350 necessary; when labeled data is scarce, fine-tuning may lead to overfitting. Various com-
 7351 binations of pre-training and fine-tuning can be employed. Pre-trained embeddings can
 7352 be used as initialization before fine-tuning, and this can substantially improve perfor-
 7353 mance (Lample et al., 2016). Alternatively, both fine-tuned and pre-trained embeddings
 7354 can be used as inputs in a single model (Kim, 2014).

7355 In semi-supervised scenarios, pretrained word embeddings can be replaced by “con-
 7356 textualized” word representations (Peters et al., 2018). These contextualized represen-
 7357 tations are set to the hidden states of a deep bi-directional LSTM, which is trained as a
 7358 bi-directional language model, motivating the name **ELMo (embeddings from language**
 7359 **models**). Given a supervised learning problem, the language model generates contextu-
 7360 alized representations, which are then used as the base layer in a task-specific supervised
 7361 neural network. This approach yields significant gains over pretrained word embeddings
 7362 on several tasks, presumably because the contextualized embeddings use unlabeled data
 7363 to learn how to integrate linguistic context into the base layer of the supervised neural
 7364 network.

7365 14.7 Distributed representations beyond distributional statistics

7366 Distributional word representations can be estimated from huge unlabeled datasets, thereby
 7367 covering many words that do not appear in labeled data: for example, GloVe embeddings
 7368 are estimated from 800 billion tokens of web data,³ while the largest labeled datasets for
 7369 NLP tasks are on the order of millions of tokens. Nonetheless, even a dataset of hundreds
 7370 of billions of tokens will not cover every word that may be encountered in the future.
 7371 Furthermore, many words will appear only a few times, making their embeddings un-
 7372 reliable. Many languages exceed English in morphological complexity, and thus have
 7373 lower token-to-type ratios. When this problem is coupled with small training corpora, it

³<http://commoncrawl.org/>

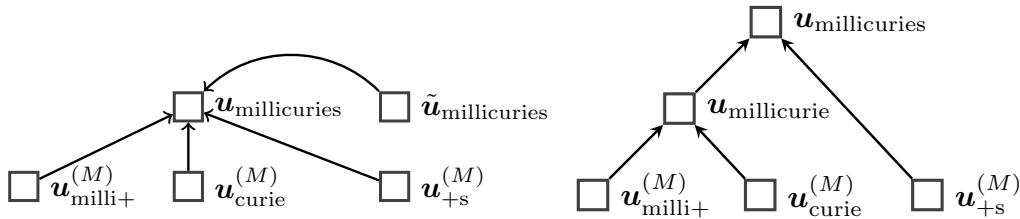


Figure 14.5: Two architectures for building word embeddings from subword units. On the left, morpheme embeddings $u^{(m)}$ are combined by addition with the non-compositional word embedding \tilde{u} (Botha and Blunsom, 2014). On the right, morpheme embeddings are combined in a recursive neural network (Luong et al., 2013).

7374 becomes especially important to leverage other sources of information beyond distributional statistics.

7376 14.7.1 Word-internal structure

7377 One solution is to incorporate word-internal structure into word embeddings. Purely
 7378 distributional approaches consider words as atomic units, but in fact, many words have
 7379 internal structure, so that their meaning can be **composed** from the representations of
 7380 sub-word units. Consider the following terms, all of which are missing from Google's
 7381 pre-trained WORD2VEC embeddings:⁴

7382 **millicuries** This word has **morphological** structure (see § 9.1.2 for more on morphology):
 7383 the prefix *milli-* indicates an amount, and the suffix *-s* indicates a plural. (A *millicurie*
 7384 is an unit of radioactivity.)

7385 **caesium** This word is a single morpheme, but the characters *-ium* are often associated
 7386 with chemical elements. (*Caesium* is the British spelling of a chemical element,
 7387 spelled *cesium* in American English.)

7388 **IAEA** This term is an acronym, as suggested by the use of capitalization. The prefix *I-* frequently
 7389 refers to international organizations, and the suffix *-A* often refers to agencies or associations. (*IAEA* is the International Atomic Energy Agency.)

7391 **Zhezhgan** This term is in title case, suggesting the name of a person or place, and the
 7392 character bigram *zh* indicates that it is likely a transliteration. (*Zhezhgan* is a mining
 7393 facility in Kazakhstan.)

⁴<https://code.google.com/archive/p/word2vec/>, accessed September 20, 2017

7394 How can word-internal structure be incorporated into word representations? One
7395 approach is to construct word representations from embeddings of the characters or mor-
7396 phemes. For example, if word i has morphological segments \mathcal{M}_i , then its embedding can
7397 be constructed by addition (Botha and Blunsom, 2014),

$$\mathbf{u}_i = \tilde{\mathbf{u}}_i + \sum_{j \in \mathcal{M}_i} \mathbf{u}_j^{(M)}, \quad [14.27]$$

7398 where $\mathbf{u}_m^{(M)}$ is a morpheme embedding and $\tilde{\mathbf{u}}_i$ is a non-compositional embedding of the
7399 whole word, which is an additional free parameter of the model (Figure 14.5, left side).
7400 All embeddings are estimated from a **log-bilinear language model** (Mnih and Hinton,
7401 2007), which is similar to the CBOW model (§ 14.5), but includes only contextual informa-
7402 tion from preceding words. The morphological segments are obtained using an unsuper-
7403 vised segmenter (Creutz and Lagus, 2007). For words that do not appear in the training
7404 data, the embedding can be constructed directly from the morphemes, assuming that each
7405 morpheme appears in some other word in the training data. The free parameter $\tilde{\mathbf{u}}$ adds
7406 flexibility: words with similar morphemes are encouraged to have similar embeddings,
7407 but this parameter makes it possible for them to be different.

7408 Word-internal structure can be incorporated into word representations in various other
7409 ways. Here are some of the main parameters.

7410 **Subword units.** Examples like *IAEA* and *Zhezghan* are not based on morphological com-
7411 position, and a morphological segmenter is unlikely to identify meaningful sub-
7412 word units for these terms. Rather than using morphemes for subword embeddings,
7413 one can use characters (Santos and Zadrozny, 2014; Ling et al., 2015; Kim et al., 2016),
7414 character n -grams (Wieting et al., 2016; Bojanowski et al., 2017), and **byte-pair en-**
7415 **codings**, a compression technique which captures frequent substrings (Gage, 1994;
7416 Sennrich et al., 2016).

7417 **Composition.** Combining the subword embeddings by addition does not differentiate
7418 between orderings, nor does it identify any particular morpheme as the **root**. A
7419 range of more flexible compositional models have been considered, including re-
7420 currence (Ling et al., 2015), convolution (Santos and Zadrozny, 2014; Kim et al.,
7421 2016), and **recursive neural networks** (Luong et al., 2013), in which representa-
7422 tions of progressively larger units are constructed over a morphological parse, e.g.
7423 $((milli+curie)+s)$, $((in+flam)+able)$, $(in+(vis+ible))$. A recursive embedding model is
7424 shown in the right panel of Figure 14.5.

7425 **Estimation.** Estimating subword embeddings from a full dataset is computationally ex-
7426 pensive. An alternative approach is to train a subword model to match pre-trained
7427 word embeddings (Cotterell et al., 2016; Pinter et al., 2017). To train such a model, it
7428 is only necessary to iterate over the vocabulary, and the not the corpus.

7429 **14.7.2 Lexical semantic resources**

Resources such as WordNet provide another source of information about word meaning; if we know that *caesium* is a synonym of *cesium*, or that a *millicurie* is a type of *measurement unit*, then this should help to provide embeddings for the unknown words, and to smooth embeddings of rare words. One way to do this is to **retrofit** pre-trained word embeddings across a network of lexical semantic relationships (Faruqui et al., 2015) by minimizing the following objective,

$$\min_{\mathbf{U}} \sum_{j=1}^V \|\mathbf{u}_i - \hat{\mathbf{u}}_i\|_2 + \sum_{(i,j) \in \mathcal{L}} \beta_{ij} \|\mathbf{u}_i - \mathbf{u}_j\|_2, \quad [14.28]$$

7430 where $\hat{\mathbf{u}}_i$ is the pretrained embedding of word i , and $\mathcal{L} = \{(i,j)\}$ is a lexicon of word
 7431 relations. The hyperparameter β_{ij} controls the importance of adjacent words having
 7432 similar embeddings; Faruqui et al. (2015) set it to the inverse of the degree of word i ,
 7433 $\beta_{ij} = |\{j : (i,j) \in \mathcal{L}\}|^{-1}$. Retrofitting improves performance on a range of intrinsic evalua-
 7434 tions, and gives small improvements on an extrinsic document classification task.

7435 **14.8 Distributed representations of multiword units**

7436 Can distributed representations extend to phrases, sentences, paragraphs, and beyond?
 7437 Before exploring this possibility, recall the distinction between distributed and distri-
 7438 butional representations. Neural embeddings such as WORD2VEC are both distributed
 7439 (vector-based) and distributional (derived from counts of words in context). As we con-
 7440 sider larger units of text, the counts decrease: in the limit, a multi-paragraph span of text
 7441 would never appear twice, except by plagiarism. Thus, the meaning of a large span of
 7442 text cannot be determined from distributional statistics alone; it must be computed com-
 7443 positionally from smaller spans. But these considerations are orthogonal to the question
 7444 of whether distributed representations — dense numerical vectors — are sufficiently ex-
 7445 pressive to capture the meaning of phrases, sentences, and paragraphs.

7446 **14.8.1 Purely distributional methods**

7447 Some multiword phrases are non-compositional: the meaning of such phrases is not de-
 7448 rived from the meaning of the individual words using typical compositional semantics.
 7449 This includes proper nouns like *San Francisco* as well as idiomatic expressions like *kick*
 7450 *the bucket* (Baldwin and Kim, 2010). For these cases, purely distributional approaches
 7451 can work. A simple approach is to identify multiword units that appear together fre-
 7452 quently, and then treat these units as words, learning embeddings using a technique such
 7453 as WORD2VEC. The problem of identifying multiword units is sometimes called **colloca-**
 7454 **tion extraction**, and can be approached using metrics such as pointwise mutual informa-
 7455 tion: two-word units are extracted first, and then larger units are extracted. Mikolov et al.

7456 (2013) identify such units and then treat them as words when estimating skipgram em-
7457 beddings, showing that the resulting embeddings perform reasonably on a task of solving
7458 phrasal analogies, e.g. *New York : New York Times :: Baltimore : Baltimore Sun*.

7459 14.8.2 Distributional-compositional hybrids

7460 To move beyond short multiword phrases, composition is necessary. A simple but sur-
7461 prisingly powerful approach is to represent a sentence with the average of its word em-
7462 beddings (Mitchell and Lapata, 2010). This can be considered a hybrid of the distribu-
7463 tional and compositional approaches to semantics: the word embeddings are computed
7464 distributionally, and then the sentence representation is computed by composition.

7465 The WORD2VEC approach can be stretched considerably further, embedding entire
7466 sentences using a model similar to skipgrams, in the “skip-thought” model of Kiros et al.
7467 (2015). Each sentence is *encoded* into a vector using a recurrent neural network: the encod-
7468 ing of sentence t is set to the RNN hidden state at its final token, $h_{M_t}^{(t)}$. This vector is then
7469 a parameter in a *decoder* model that is used to generate the previous and subsequent sen-
7470 tences: the decoder is another recurrent neural network, which takes the encoding of the
7471 neighboring sentence as an additional parameter in its recurrent update. (This **encoder-**
7472 **decoder model** is discussed at length in chapter 18.) The encoder and decoder are trained
7473 simultaneously from a likelihood-based objective, and the trained encoder can be used to
7474 compute a distributed representation of any sentence. Skip-thought can also be viewed
7475 as a hybrid of distributional and compositional approaches: the vector representation of
7476 each sentence is computed compositionally from the representations of the individual
7477 words, but the training objective is distributional, based on sentence co-occurrence across
7478 a corpus.

7479 **Autoencoders** are a variant of encoder-decoder models in which the decoder is trained
7480 to produce the same text that was originally encoded, using only the distributed encod-
7481 ing vector (Li et al., 2015). The encoding acts as a bottleneck, so that generalization is
7482 necessary if the model is to successfully fit the training data. In **denoising autoencoders**,
7483 the input is a corrupted version of the original sentence, and the auto-encoder must re-
7484 construct the uncorrupted original (Vincent et al., 2010; Hill et al., 2016). By interpolating
7485 between distributed representations of two sentences, $\alpha \mathbf{u}_i + (1 - \alpha) \mathbf{u}_j$, it is possible to gen-
7486 erate sentences that combine aspects of the two inputs, as shown in Figure 14.6 (Bowman
7487 et al., 2016).

7488 Autoencoders can also be applied to longer texts, such as paragraphs and documents.
7489 This enables applications such as **question answering**, which can be performed by match-
7490 ing the encoding of the question with encodings of candidate answers (Miao et al., 2016).

this was the only way
 it was the only way
 it was her turn to blink
 it was hard to tell
 it was time to move on
 he had to do it again
 they all looked at each other
 they all turned to look back
 they both turned to face him
they both turned and walked away

Figure 14.6: By interpolating between the distributed representations of two sentences (in bold), it is possible to generate grammatical sentences that combine aspects of both (Bowman et al., 2016)

7491 14.8.3 Supervised compositional methods

7492 Given a supervision signal, such as a label describing the sentiment or meaning of a sen-
 7493 tence, a wide range of compositional methods can be applied to compute a distributed
 7494 representation that then predicts the label. The simplest is to average the embeddings
 7495 of each word in the sentence, and pass this average through a feedforward neural net-
 7496 work (Iyyer et al., 2015). Convolutional and recurrent neural networks go further, with
 7497 the ability to effectively capturing multiword phenomena such as negation (Kalchbrenner
 7498 et al., 2014; Kim, 2014; Li et al., 2015; Tang et al., 2015). Another approach is to incorpo-
 7499 rate the syntactic structure of the sentence into a **recursive neural networks**, in which the
 7500 representation for each syntactic constituent is computed from the representations of its
 7501 children (Socher et al., 2012). However, in many cases, recurrent neural networks perform
 7502 as well or better than recursive networks (Li et al., 2015).

7503 Whether convolutional, recurrent, or recursive, a key question is whether supervised
 7504 sentence representations are task-specific, or whether a single supervised sentence repre-
 7505 sentation model can yield useful performance on other tasks. Wieting et al. (2015) train a
 7506 variety of sentence embedding models for the task of labeling pairs of sentences as **para-**
 7507 **phrases**. They show that the resulting sentence embeddings give good performance for
 7508 sentiment analysis. The **Stanford Natural Language Inference corpus** classifies sentence
 7509 pairs as **entailments** (the truth of sentence i implies the truth of sentence j), **contradictions**
 7510 (the truth of sentence i implies the falsity of sentence j), and neutral (i neither entails nor
 7511 contradicts j). Sentence embeddings trained on this dataset transfer to a wide range of
 7512 classification tasks (Conneau et al., 2017).

7513 14.8.4 Hybrid distributed-symbolic representations

7514 The power of distributed representations is in their generality: the distributed represen-
7515 tation of a unit of text can serve as a summary of its meaning, and therefore as the input
7516 for downstream tasks such as classification, matching, and retrieval. For example, dis-
7517 tributed sentence representations can be used to recognize the paraphrase relationship
7518 between closely related sentences like the following:

- 7519 (14.5) Donald thanked Vlad profusely.
7520 (14.6) Donald conveyed to Vlad his profound appreciation.
7521 (14.7) Vlad was showered with gratitude by Donald.

7522 Symbolic representations are relatively brittle to this sort of variation, but are better
7523 suited to describe individual entities, the things that they do, and the things that are done
7524 to them. In examples (14.5)-(14.7), we not only know that somebody thanked someone
7525 else, but we can make a range of inferences about what has happened between the en-
7526 tities named *Donald* and *Vlad*. Because distributed representations do not treat entities
7527 symbolically, they lack the ability to reason about the roles played by entities across a sen-
7528 tence or larger discourse.⁵ A hybrid between distributed and symbolic representations
7529 might give the best of both worlds: robustness to the many different ways of describing
7530 the same event, plus the expressiveness to support inferences about entities and the roles
7531 that they play.

7532 A “top-down” hybrid approach is to begin with logical semantics (of the sort de-
7533 scribed in the previous two chapters), and but replace the predefined lexicon with a set
7534 of distributional word clusters (Poon and Domingos, 2009; Lewis and Steedman, 2013). A
7535 “bottom-up” approach is to add minimal symbolic structure to existing distributed repre-
7536 sentations, such as vector representations for each entity (Ji and Eisenstein, 2015; Wiseman
7537 et al., 2016). This has been shown to improve performance on two problems that we will
7538 encounter in the following chapters: classification of **discourse relations** between adjac-
7539 ent sentences (chapter 16; Ji and Eisenstein, 2015), and **coreference resolution** of entity
7540 mentions (chapter 15; Wiseman et al., 2016; Ji et al., 2017). Research on hybrid seman-
7541 tic representations is still in an early stage, and future representations may deviate more
7542 boldly from existing symbolic and distributional approaches.

7543 Additional resources

7544 Turney and Pantel (2010) survey a number of facets of vector word representations, fo-
7545 cusing on matrix factorization methods. Schnabel et al. (2015) highlight problems with

⁵At a 2014 workshop on semantic parsing, this critique of distributed representations was expressed by Ray Mooney — a leading researcher in computational semantics — in a now well-known quote, “you can’t cram the meaning of a whole sentence into a single vector!”

7546 similarity-based evaluations of word embeddings, and present a novel evaluation that
 7547 controls for word frequency. Baroni et al. (2014) address linguistic issues that arise in
 7548 attempts to combine distributed and compositional representations.

7549 In bilingual and multilingual distributed representations, embeddings are estimated
 7550 for translation pairs or tuples, such as (*dog, perro, chien*). These embeddings can improve
 7551 machine translation (Zou et al., 2013; Klementiev et al., 2012), transfer natural language
 7552 processing models across languages (Täckström et al., 2012), and make monolingual word
 7553 embeddings more accurate (Faruqui and Dyer, 2014). A typical approach is to learn a pro-
 7554 jection that maximizes the correlation of the distributed representations of each element
 7555 in a translation pair, which can be obtained from a bilingual dictionary. Distributed rep-
 7556 resentations can also be linked to perceptual information, such as image features. Bruni
 7557 et al. (2014) use textual descriptions of images to obtain visual contextual information for
 7558 various words, which supplements traditional distributional context. Image features can
 7559 also be inserted as contextual information in log bilinear language models (Kiros et al.,
 7560 2014), making it possible to automatically generate text descriptions of images.

7561 Exercises

- 7562 1. Prove that the sum of probabilities of paths through a hierarchical softmax tree is
 7563 equal to one.
- 7564 2. In skipgram word embeddings, the negative sampling objective can be written as,

$$\mathcal{L} = \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{C}} \text{count}(i, j) \psi(i, j), \quad [14.29]$$

7565 with $\psi(i, j)$ is defined in Equation 14.23.

7566 Suppose we draw the negative samples from the empirical unigram distribution
 $\hat{p}(i) = p_{\text{unigram}}(i)$. First, compute the expectation of \mathcal{L} with respect this probability.

7567 Next, take the derivative of this expectation with respect to the score of a single word
 7568 context pair $\sigma(\mathbf{u}_i \cdot \mathbf{v}_j)$, and solve for the pointwise mutual information $\text{PMI}(i, j)$. You
 7569 should be able to show that at the optimum, the PMI is a simple function of $\sigma(\mathbf{u}_i \cdot \mathbf{v}_j)$
 7570 and the number of negative samples.

- 7571 3. * In Brown clustering, prove that the cluster merge that maximizes the average mu-
 7572 tual information (Equation 14.13) also maximizes the log-likelihood objective (Equa-
 7573 tion 14.12).

7574 For the next two problems, download a set of pre-trained word embeddings, such as the
 7575 WORD2VEC or polyglot embeddings.

7576 4. Use cosine similarity to find the most similar words to: *dog, whale, before, however,*
7577 *fabricate.*

7578 5. Use vector addition and subtraction to compute target vectors for the analogies be-
7579 low. After computing each target vector, find the top three candidates by cosine
7580 similarity.

- 7581 • *dog:puppy :: cat: ?*
7582 • *speak:speaker :: sing:?*
7583 • *France:French :: England:?*
7584 • *France:wine :: England:?*

7585 The remaining problems will require you to build a classifier and test its properties. Pick
7586 a multi-class text classification dataset, such as RCV1⁶). Divide your data into training
7587 (60%), development (20%), and test sets (20%), if no such division already exists.

7588 6. Train a convolutional neural network, with inputs set to pre-trained word embed-
7589 dings from the previous problem. Use a special, fine-tuned embedding for out-of-
7590 vocabulary words. Train until performance on the development set does not im-
7591 prove. You can also use the development set to tune the model architecture, such
7592 as the convolution width and depth. Report *F-MEASURE* and accuracy, as well as
7593 training time.

7594 7. Now modify your model from the previous problem to fine-tune the word embed-
7595 dings. Report *F-MEASURE*, accuracy, and training time.

7596 8. Try a simpler approach, in which word embeddings in the document are averaged,
7597 and then this average is passed through a feed-forward neural network. Again, use
7598 the development data to tune the model architecture. How close is the accuracy to
7599 the convolutional networks from the previous problems?

⁶http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm

7600 Chapter 15

7601 Reference Resolution

7602 References are one of the most noticeable forms of linguistic ambiguity, afflicting not just
7603 automated natural language processing systems, but also fluent human readers. Warnings
7604 to avoid “ambiguous pronouns” are ubiquitous in manuals and tutorials on writing
7605 style. But referential ambiguity is not limited to pronouns, as shown in the text in Fig-
7606 ure 15.1. Each of the bracketed substrings refers to an entity that is introduced earlier
7607 in the passage. These references include the pronouns *he* and *his*, but also the shortened
7608 name *Cook*, and **nominals** such as *the firm* and *the firm’s biggest growth market*.

7609 **Reference resolution** subsumes several subtasks. This chapter will focus on **corefer-
7610 ence resolution**, which is the task of grouping spans of text that refer to a single underly-
7611 ing entity, or, in some cases, a single event: for example, the spans *Tim Cook*, *he*, and *Cook*
7612 are all **coreferent**. These individual spans are called **mentions**, because they mention an
7613 entity; the entity is sometimes called the **referent**. Each mention has a set of **antecedents**,
7614 which are preceding mentions that are coreferent; for the first mention of an entity, the an-
7615 tecedent set is empty. The task of **pronominal anaphora resolution** requires identifying
7616 only the antecedents of pronouns. In **entity linking**, references are resolved not to other
7617 spans of text, but to entities in a knowledge base. This task is discussed in chapter 17.

7618 Coreference resolution is a challenging problem for several reasons. Resolving differ-
7619 ent types of **referring expressions** requires different types of reasoning: the features and
7620 methods that are useful for resolving pronouns are different from those that are useful
7621 to resolve names and nominals. Coreference resolution involves not only linguistic rea-
7622 soning, but also world knowledge and pragmatics: you may not have known that China
7623 was Apple’s biggest growth market, but it is likely that you effortlessly resolved this ref-
7624 erence while reading the passage in Figure 15.1.¹ A further challenge is that coreference

¹This interpretation is based in part on the assumption that a **cooperative** author would not use the expression *the firm’s biggest growth market* to refer to an entity not yet mentioned in the article (Grice, 1975). **Pragmatics** is the discipline of linguistics concerned with the formalization of such assumptions (Huang,

- (15.1) *[[Apple Inc] Chief Executive Tim Cook] has jetted into [China] for talks with government officials as [he] seeks to clear up a pile of problems in [[the firm] 's biggest growth market] ... [Cook] is on [his] first trip to [the country] since taking over...*

Figure 15.1: Running example (Yee and Jones, 2012). Coreferring entity mentions are underlined and bracketed.

7625 resolution decisions are often entangled: each mention adds information about the entity,
 7626 which affects other coreference decisions. This means that coreference resolution must
 7627 be addressed as a structure prediction problem. But as we will see, there is no dynamic
 7628 program that allows the space of coreference decisions to be searched efficiently.

7629 15.1 Forms of referring expressions

7630 There are three main forms of referring expressions — pronouns, names, and nominals.

7631 15.1.1 Pronouns

7632 Pronouns are a closed class of words that are used for references. A natural way to think
 7633 about pronoun resolution is SMASH (Kehler, 2007):

- 7634 • Search for candidate antecedents;
- 7635 • Match against hard agreement constraints;
- 7636 • And Select using Heuristics, which are “soft” constraints such as recency, syntactic
 7637 prominence, and parallelism.

7638 15.1.1.1 Search

7639 In the search step, candidate antecedents are identified from the preceding text or speech.²
 7640 Any noun phrase can be a candidate antecedent, and pronoun resolution usually requires

2015).

²Pronouns whose referents come later are known as **cataphora**, as in this example from Márquez (1970):

- (15.1) Many years later, as [he] faced the firing squad, [Colonel Aureliano Buendía] was to remember that distant afternoon when his father took him to discover ice.

7641 parsing the text to identify all such noun phrases.³ Filtering heuristics can help to prune
 7642 the search space to noun phrases that are likely to be coreferent (Lee et al., 2013; Durrett
 7643 and Klein, 2013). In nested noun phrases, mentions are generally considered to be the
 7644 largest unit with a given head word: thus, *Apple Inc. Chief Executive Tim Cook* would be
 7645 included as a mention, but *Tim Cook* would not, since they share the same head word,
 7646 *Cook*.

7647 15.1.1.2 Matching constraints for pronouns

7648 References and their antecedents must agree on semantic features such as number, person,
 7649 gender, and animacy. Consider the pronoun *he* in this passage from the running example:

7650 (15.2) Tim Cook has jetted in for talks with officials as [he] seeks to clear up a pile of
 7651 problems...

7652 The pronoun and possible antecedents have the following features:

- 7653 • *he*: singular, masculine, animate, third person
- 7654 • *officials*: plural, animate, third person
- 7655 • *talks*: plural, inanimate, third person
- 7656 • *Tim Cook*: singular, masculine, animate, third person

7657 The SMASH method searches backwards from *he*, discarding *officials* and *talks* because they
 7658 do not satisfy the agreements constraints.

7659 Another source of constraints comes from syntax — specifically, from the phrase struc-
 7660 ture trees discussed in chapter 10. Consider a parse tree in which both *x* and *y* are phrasal
 7661 constituents. The constituent *x* **c-commands** the constituent *y* iff the first branching node
 7662 above *x* also dominates *y*. For example, in Figure 15.2a, *Abigail* c-commands *her*, because
 7663 the first branching node above *Abigail*, *S*, also dominates *her*. Now, if *x* c-commands *y*,
 7664 **government and binding theory** (Chomsky, 1982) states that *y* can refer to *x* only if it is
 7665 a **reflexive pronoun** (e.g., *herself*). Furthermore, if *y* is a reflexive pronoun, then its an-
 7666 tecedent must c-command it. Thus, in Figure 15.2a, *her* cannot refer to *Abigail*; conversely,
 7667 if we replace *her* with *herself*, then the reflexive pronoun *must* refer to *Abigail*, since this is
 7668 the only candidate antecedent that c-commands it.

7669 Now consider the example shown in Figure 15.2b. Here, *Abigail* does not c-command
 7670 *her*, but *Abigail's mom* does. Thus, *her* can refer to *Abigail* — and we cannot use reflexive

³In the OntoNotes coreference annotations, verbs can also be antecedents, if they are later referenced by nominals (Pradhan et al., 2011):

(15.1) Sales of passenger cars [grew] 22%. [The strong growth] followed year-to-year increases.

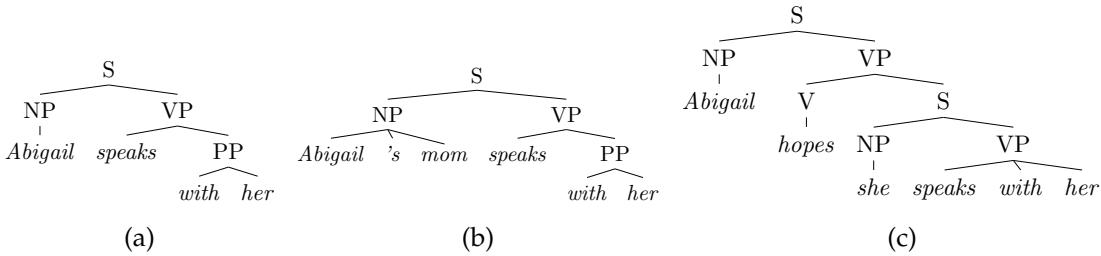


Figure 15.2: In (a), *Abigail* c-commands *her*; in (b), *Abigail* does not c-command *her*, but *Abigail's mom* does; in (c), the scope of *Abigail* is limited by the S non-terminal, so that *she* or *her* can bind to *Abigail*, but not both.

7671 *herself* in this context, unless we are talking about *Abigail*'s mom. However, *her* does not
 7672 have to refer to *Abigail*. Finally, Figure 15.2c shows how these constraints are limited.
 7673 In this case, the pronoun *she* can refer to *Abigail*, because the S non-terminal puts *Abigail*
 7674 outside the domain of *she*. Similarly, *her* can also refer to *Abigail*. But *she* and *her* cannot be
 7675 coreferent, because *she* c-commands *her*.

7676 15.1.1.3 Heuristics

7677 After applying constraints, heuristics are applied to select among the remaining candidates.
 7678 Recency is a particularly strong heuristic. All things equal, readers will prefer
 7679 the more recent referent for a given pronoun, particularly when comparing referents that
 7680 occur in different sentences. Jurafsky and Martin (2009) offer the following example:

- 7681 (15.3) The doctor found an old map in the captain's chest. Jim found an even older map
 7682 hidden on the shelf. [It] described an island.

7683 Readers are expected to prefer the older map as the referent for the pronoun *it*.

7684 However, subjects are often preferred over objects, and this can contradict the preference
 7685 for recency when two candidate referents are in the same sentence. For example,

- 7686 (15.4) Asha loaned Mei a book on Spanish. [She] is always trying to help people.

7687 Here, we may prefer to link *she* to *Asha* rather than *Mei*, because of *Asha*'s position in the
 7688 subject role of the preceding sentence. (Arguably, this preference would not be strong
 7689 enough to select *Asha* if the second sentence were *She is visiting Valencia next month*.)

7690 A third heuristic is parallelism:

- 7691 (15.5) Asha loaned Mei a book on Spanish. Olya loaned [her] a book on Portuguese.

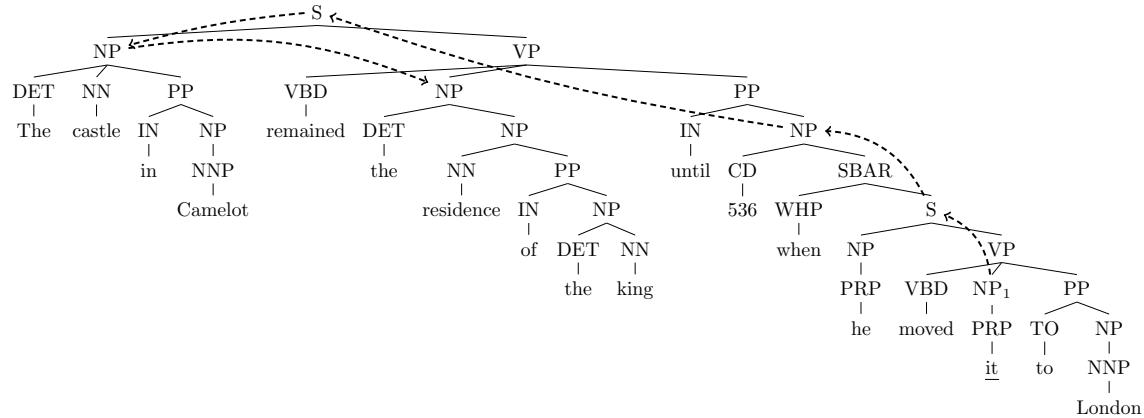


Figure 15.3: Left-to-right breadth-first tree traversal (Hobbs, 1978), indicating that the search for an antecedent for *it* (NP₁) would proceed in the following order: 536; *the castle in Camelot*; *the residence of the king*; *Camelot*; *the king*. Hobbs (1978) proposes semantic constraints to eliminate 536 and *the castle in Camelot* as candidates, since they are unlikely to be the direct object of the verb *move*.

7692 Here *Mei* is preferred as the referent for *her*, contradicting the preference for the subject
 7693 *Asha* in the preceding sentence.

7694 The recency and subject role heuristics can be unified by traversing the document in
 7695 a syntax-driven fashion (Hobbs, 1978): each preceding sentence is traversed breadth-first,
 7696 left-to-right (Figure 15.3). This heuristic successfully handles (15.4): *Asha* is preferred as
 7697 the referent for *she* because the subject NP is visited first. It also handles (15.3): the older
 7698 map is preferred as the referent for *it* because the more recent sentence is visited first. (An
 7699 alternative unification of recency and syntax is proposed by **centering theory** (Grosz et al.,
 7700 1995), which is discussed in detail in chapter 16.)

7701 In early work on reference resolution, the number of heuristics was small enough that
 7702 a set of numerical weights could be set by hand (Lappin and Leass, 1994). More recent
 7703 work uses machine learning to quantify the importance of each of these factors. However,
 7704 pronoun resolution cannot be completely solved by constraints and heuristics alone. This
 7705 is shown by the classic example pair (Winograd, 1972):

7706 (15.6) The [city council] denied [the protesters] a permit because [they] advocated / feared
 7707 violence.

7708 Without reasoning about the motivations of the city council and protesters, it is unlikely
 7709 that any system could correctly resolve both versions of this example.

7710 **15.1.1.4 Non-referential pronouns**

7711 While pronouns are generally used for reference, they need not refer to entities. The fol-
7712 lowing examples show how pronouns can refer to propositions, events, and speech acts.

- 7713 (15.7) They told me that I was too ugly for show business, but I didn't believe [it].
7714 (15.8) Asha saw Babak get angry, and I saw [it] too.
7715 (15.9) Asha said she worked in security. I suppose [that]'s one way to put it.

7716 These forms of reference are generally not annotated in large-scale coreference resolution
7717 datasets such as OntoNotes (Pradhan et al., 2011).

7718 Pronouns may also have **generic referents**:

- 7719 (15.10) A poor carpenter blames [her] tools.
7720 (15.11) On the moon, [you] have to carry [your] own oxygen.
7721 (15.12) Every farmer who owns a donkey beats [it]. (Geach, 1962)

7722 In the OntoNotes dataset, coreference is not annotated for generic referents, even in cases
7723 like these examples, in which the same generic entity is mentioned multiple times.

7724 Some pronouns do not refer to anything at all:

- 7725 (15.13) *[It]'s raining.*
[Il] pleut. (Fr)
7726 (15.14) [It] 's money that she's really after.
7727 (15.15) [It] is too bad that we have to work so hard.

7728 How can we automatically distinguish these usages of *it* from referential pronouns?
7729 Consider the the difference between the following two examples (Bergsma et al., 2008):

- 7730 (15.16) You can make [it] in advance.
7731 (15.17) You can make [it] in showbiz.

7732 In the second example, the pronoun *it* is non-referential. One way to see this is by substi-
7733 tuting another pronoun, like *them*, into these examples:

- 7734 (15.18) You can make [them] in advance.
7735 (15.19) ? You can make [them] in showbiz.

7736 The questionable grammaticality of the second example suggests that *it* is not referential.
7737 Bergsma et al. (2008) operationalize this idea by comparing distributional statistics for the

7738 *n*-grams around the word *it*, testing how often other pronouns or nouns appear in the
7739 same context. In cases where nouns and other pronouns are infrequent, the *it* is unlikely
7740 to be referential.

7741 15.1.2 Proper Nouns

7742 If a proper noun is used as a referring expression, it often corefers with another proper
7743 noun, so that the coreference problem is simply to determine whether the two names
7744 match. Subsequent proper noun references often use a shortened form, as in the running
7745 example (Figure 15.1):

7746 (15.20) Apple Inc Chief Executive [Tim Cook] has jetted into China ... [Cook] is on his
7747 first business trip to the country ...

7748 A typical solution for proper noun coreference is to match the syntactic **head words**
7749 of the reference with the referent. In § 10.5.2, we saw that the head word of a phrase can
7750 be identified by applying head percolation rules to the phrasal parse tree; alternatively,
7751 the head can be identified as the root of the dependency subtree covering the name. For
7752 sequences of proper nouns, the head word will be the final token.

7753 There are a number of caveats to the practice of matching head words of proper nouns.

- 7754 • In the European tradition, family names tend to be more specific than given names,
7755 and family names usually come last. However, other traditions have other practices:
7756 for example, in Chinese names, the family name typically comes first; in Japanese,
7757 honorifics come after the name, as in *Nobu-San* (*Mr. Nobu*).
- 7758 • In organization names, the head word is often not the most informative, as in *Georgia*
7759 *Tech* and *Virginia Tech*. Similarly, *Lebanon* does not refer to the same entity as *Southern Lebanon*, necessitating special rules for the specific case of geographical modi-
7760 fiers (Lee et al., 2011).
- 7762 • Proper nouns can be nested, as in [*the CEO of [Microsoft]*], resulting in head word
7763 match without coreference.

7764 Despite these difficulties, proper nouns are the easiest category of references to re-
7765 solve (Stoyanov et al., 2009). In machine learning systems, one solution is to include a
7766 range of matching features, including exact match, head match, and string inclusion. In
7767 addition to matching features, competitive systems (e.g., Bengtson and Roth, 2008) in-
7768 clude large lists, or **gazetteers**, of acronyms (e.g., *the National Basketball Association/NBA*),
7769 demonymns (e.g., *the Israelis/Israel*), and other aliases (e.g., *the Georgia Institute of Technol-*
7770 *ogy/Georgia Tech*).

7771 **15.1.3 Nominals**

7772 In coreference resolution, noun phrases that are neither pronouns nor proper nouns are
 7773 referred to as **nominals**. In the running example (Figure 15.1), nominal references include:
 7774 *the firm (Apple Inc); the firm's biggest growth market (China); and the country (China)*.

7775 Nominals are especially difficult to resolve (Denis and Baldridge, 2007; Durrett and
 7776 Klein, 2013), and the examples above suggest why this may be the case: world knowledge
 7777 is required to identify *Apple Inc* as a *firm*, and *China* as a *growth market*. Other difficult
 7778 examples include the use of colloquial expressions, such as coreference between *Clinton*
 7779 *campaign officials* and *the Clinton camp* (Soon et al., 2001).

7780 **15.2 Algorithms for coreference resolution**

The ground truth training data for coreference resolution is a set of mention sets, where all mentions within each set refer to a single entity.⁴ In the running example from Figure 15.1, the ground truth coreference annotation is:

$$c_1 = \{Apple\ Inc_{1:2}, the\ firm_{27:28}\} \quad [15.1]$$

$$c_2 = \{Apple\ Inc\ Chief\ Executive\ Tim\ Cook_{1:6}, he_{17}, Cook_{33}, his_{36}\} \quad [15.2]$$

$$c_3 = \{China_{10}, the\ firm\ 's\ biggest\ growth\ market_{27:32}, the\ country_{40:41}\} \quad [15.3]$$

7781 Each row specifies the token spans that mention an entity. (“Singleton” entities, which are
 7782 mentioned only once (e.g., *talks, government officials*), are excluded from the annotations.)
 7783 Equivalently, if given a set of M mentions, $\{m_i\}_{i=1}^M$, each mention i can be assigned to a
 7784 cluster z_i , where $z_i = z_j$ if i and j are coreferent. The cluster assignments z are invariant
 7785 under permutation. The unique clustering associated with the assignment z is written
 7786 $c(z)$.

7787 **Mention identification** The task of identifying mention spans for coreference resolution
 7788 is often performed by applying a set of heuristics to the phrase structure parse of each
 7789 sentence. A typical approach is to start with all noun phrases and named entities, and
 7790 then apply filtering rules to remove nested noun phrases with the same head (e.g., [*Apple*
 7791 *CEO [Tim Cook]*]), numeric entities (e.g., [*100 miles*], [*97%*]), non-referential *it*, etc (Lee
 7792 et al., 2013; Durrett and Klein, 2013). In general, these deterministic approaches err in
 7793 favor of recall, since the mention clustering component can choose to ignore false positive
 7794 mentions, but cannot recover from false negatives. An alternative is to consider all spans

⁴In many annotations, the term **markable** is used to refer to spans of text that can *potentially* mention an entity. The set of markables includes non-referential pronouns, which does not mention any entity. Part of the job of the coreference system is to avoid incorrectly linking these non-referential markables to any mention chains.

7795 (up to some finite length) as candidate mentions, performing mention identification and
 7796 clustering jointly (Daumé III and Marcu, 2005; Lee et al., 2017).

7797 **Mention clustering** The overwhelming majority of research on coreference resolution
 7798 addresses the subtask of mention clustering, and this will be the focus of the remainder of
 7799 this chapter. There are two main sets of approaches. In *mention-based models*, the scoring
 7800 function for a coreference clustering decomposes over pairs of mentions. These pairwise
 7801 decisions are then aggregated, using a clustering heuristic. Mention-based coreference
 7802 clustering can be treated as a fairly direct application of supervised classification or rank-
 7803 ing. However, the mention-pair locality assumption can result in incoherent clusters, like
 7804 $\{Hillary Clinton \leftarrow Clinton \leftarrow Mr Clinton\}$, in which the pairwise links score well, but the
 7805 overall result is unsatisfactory. *Entity-based models* address this issue by scoring entities
 7806 holistically. This can make inference more difficult, since the number of possible entity
 7807 groupings is exponential in the number of mentions.

7808 15.2.1 Mention-pair models

7809 In the **mention-pair model**, a binary label $y_{i,j} \in \{0, 1\}$ is assigned to each pair of mentions
 7810 (i, j) , where $i < j$. If i and j corefer ($z_i = z_j$), then $y_{i,j} = 1$; otherwise, $y_{i,j} = 0$. The
 7811 mention *he* in Figure 15.1 is preceded by five other mentions: (1) *Apple Inc*; (2) *Apple Inc*
 7812 *Chief Executive Tim Cook*; (3) *China*; (4) *talks*; (5) *government officials*. The correct mention
 7813 pair labeling is $y_{2,6} = 1$ and $y_{i \neq 2,6} = 0$ for all other i . If a mention j introduces a new entity,
 7814 such as mention 3 in the example, then $y_{i,j} = 0$ for all i . The same is true for “mentions”
 7815 that do not refer to any entity, such as non-referential pronouns. If mention j refers to an
 7816 entity that has been mentioned more than once, then $y_{i,j} = 1$ for all $i < j$ that mention the
 7817 referent.

7818 By transforming coreference into a set of binary labeling problems, the mention-pair
 7819 model makes it possible to apply an off-the-shelf binary classifier (Soon et al., 2001). This
 7820 classifier is applied to each mention j independently, searching backwards from j until
 7821 finding an antecedent i which corefers with j with high confidence. After identifying a
 7822 single **antecedent**, the remaining mention pair labels can be computed by transitivity: if
 7823 $y_{i,j} = 1$ and $y_{j,k} = 1$, then $y_{i,k} = 1$.

7824 Since the ground truth annotations give entity chains c but not individual mention-
 7825 pair labels y , an additional heuristic must be employed to convert the labeled data into
 7826 training examples for classification. A typical approach is to generate at most one pos-
 7827 itive labeled instance $y_{a_j,j} = 1$ for mention j , where a_j is the index of the most recent
 7828 antecedent, $a_j = \max\{i : i < j \wedge z_i = z_j\}$. Negative labeled instances are generated for
 7829 all for all $i \in \{a_j + 1, \dots, j\}$. In the running example, the most recent antecedent of the
 7830 pronoun *he* is $a_6 = 2$, so the training data would be $y_{2,6} = 1$ and $y_{3,6} = y_{4,6} = y_{5,6} = 0$.

7831 The variable $y_{1,6}$ is not part of the training data, because the first mention appears before
 7832 the true antecedent $a_6 = 2$.

7833 **15.2.2 Mention-ranking models**

In **mention ranking** (Denis and Baldridge, 2007), the classifier learns to identify a single antecedent $a_i \in \{\epsilon, 1, 2, \dots, i-1\}$ for each referring expression i ,

$$\hat{a}_i = \operatorname{argmax}_{a \in \{\epsilon, 1, 2, \dots, i-1\}} \psi_M(a, i), \quad [15.4]$$

7834 where $\psi_M(a, i)$ is a score for the mention pair (a, i) . If $a = \epsilon$, then mention i does not refer
 7835 to any previously-introduced entity — it is not **anaphoric**. Mention-ranking is similar to
 7836 the mention-pair model, but all candidates are considered simultaneously, and at most
 7837 a single antecedent is selected. The mention-ranking model explicitly accounts for the
 7838 possibility that mention i is not anaphoric, through the score $\psi_M(\epsilon, i)$. The determination
 7839 of anaphoricity can be made by a special classifier in a preprocessing step, so that non- ϵ
 7840 antecedents are identified only for spans that are determined to be anaphoric (Denis and
 7841 Baldridge, 2008).

7842 As a learning problem, ranking can be trained using the same objectives as in dis-
 7843 criminative classification. For each mention i , we can define a gold antecedent a_i^* , and an
 7844 associated loss, such as the hinge loss, $\ell_i = (1 - \psi_M(a_i^*, i) + \psi_M(\hat{a}, i))_+$ or the negative
 7845 log-likelihood, $\ell_i = -\log p(a_i^* | i; \theta)$. (For more on learning to rank, see § 17.1.1.) But as
 7846 with the mention-pair model, there is a mismatch between the labeled data, which comes
 7847 in the form of mention sets, and the desired supervision, which would indicate the spe-
 7848 cific antecedent of each mention. The antecedent variables $\{a_i\}_{i=1}^M$ relate to the mention
 7849 sets in a many-to-one mapping: each set of antecedents induces a single clustering, but a
 7850 clustering can correspond to many different settings of antecedent variables.

A heuristic solution is to set $a_i^* = \max\{j : j < i \wedge z_j = z_i\}$, the most recent mention in
 the same cluster as i . But the most recent mention may not be the most informative: in the
 running example, the most recent antecedent of the mention *Cook* is the pronoun *he*, but
 a more useful antecedent is the earlier mention *Apple Inc Chief Executive Tim Cook*. Rather
 than selecting a specific antecedent to train on, the antecedent can be treated as a latent
 variable, in the manner of the **latent variable perceptron** from § 12.4.2 (Fernandes et al.,

2014):

$$\hat{\mathbf{a}} = \operatorname{argmax}_{\mathbf{a}} \sum_{i=1}^M \psi_M(a_i, i) \quad [15.5]$$

$$\mathbf{a}^* = \operatorname{argmax}_{\mathbf{a} \in \mathcal{A}(c)} \sum_{i=1}^M \psi_M(a_i, i) \quad [15.6]$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \sum_{i=1}^M \frac{\partial L}{\partial \boldsymbol{\theta}} \psi_M(a_i^*, i) - \sum_{i=1}^M \frac{\partial L}{\partial \boldsymbol{\theta}} \psi_M(\hat{a}_i, i) \quad [15.7]$$

where $\mathcal{A}(c)$ is the set of antecedent structures that is compatible with the ground truth coreference clustering c . Another alternative is to sum over all the conditional probabilities of antecedent structures that are compatible with the ground truth clustering (Durrett and Klein, 2013; Lee et al., 2017). For the set of mention \mathbf{m} , we compute the following probabilities:

$$p(c | \mathbf{m}) = \sum_{\mathbf{a} \in \mathcal{A}(c)} p(\mathbf{a} | \mathbf{m}) = \sum_{\mathbf{a} \in \mathcal{A}(c)} \prod_{i=1}^M p(a_i | i, \mathbf{m}) \quad [15.8]$$

$$p(a_i | i, \mathbf{m}) = \frac{\exp(\psi_M(a_i, i))}{\sum_{a' \in \{\epsilon, 1, 2, \dots, i-1\}} \exp(\psi_M(a', i))}. \quad [15.9]$$

7851 This objective rewards models that assign high scores for all valid antecedent structures.
 7852 In the running example, this would correspond to summing the probabilities of the two
 7853 valid antecedents for *Cook, he* and *Apple Inc Chief Executive Tim Cook*. In one of the exer-
 7854 cises, you will compute the number of valid antecedent structures for a given clustering.

7855 15.2.3 Transitive closure in mention-based models

A problem for mention-based models is that individual mention-level decisions may be incoherent. Consider the following mentions:

$$m_1 = \text{Hillary Clinton} \quad [15.10]$$

$$m_2 = \text{Clinton} \quad [15.11]$$

$$m_3 = \text{Bill Clinton} \quad [15.12]$$

7856 A mention-pair system might predict $\hat{y}_{1,2} = 1, \hat{y}_{2,3} = 1, \hat{y}_{1,3} = 0$. Similarly, a mention-
 7857 ranking system might choose $\hat{a}_2 = 1$ and $\hat{a}_3 = 2$. Logically, if mentions 1 and 3 are both
 7858 coreferent with mention 2, then all three mentions must refer to the same entity. This
 7859 constraint is known as **transitive closure**.

Transitive closure can be applied *post hoc*, revising the independent mention-pair or mention-ranking decisions. However, there are many possible ways to enforce transitive closure: in the example above, we could set $\hat{y}_{1,3} = 1$, or $\hat{y}_{1,2} = 0$, or $\hat{y}_{2,3} = 0$. For documents with many mentions, there may be many violations of transitive closure, and many possible fixes. Transitive closure can be enforced by always adding edges, so that $\hat{y}_{1,3} = 1$ is preferred (e.g., Soon et al., 2001), but this can result in overclustering, with too many mentions grouped into too few entities.

Mention-pair coreference resolution can be viewed as a constrained optimization problem,

$$\begin{aligned} \max_{\mathbf{y} \in \{0,1\}^M} \quad & \sum_{j=1}^M \sum_{i=1}^j \psi_M(i, j) \times y_{i,j} \\ \text{s.t.} \quad & y_{i,j} + y_{j,k} - 1 \leq y_{i,k}, \quad \forall i < j < k, \end{aligned}$$

with the constraint enforcing transitive closure. This constrained optimization problem is equivalent to graph partitioning with positive and negative edge weights: construct a graph where the nodes are mentions, and the edges are the pairwise scores $\psi_M(i, j)$; the goal is to partition the graph so as to maximize the sum of the edge weights between all nodes within the same partition (McCallum and Wellner, 2004). This problem is NP-hard, motivating approximations such as correlation clustering (Bansal et al., 2004) and **integer linear programming** (Klenner, 2007; Finkel and Manning, 2008, also see § 13.2.2).

15.2.4 Entity-based models

A weakness of mention-based models is that they treat coreference resolution as a classification or ranking problem, when it is really a clustering problem: the goal is to group the mentions together into clusters that correspond to the underlying entities. Entity-based approaches attempt to identify these clusters directly. Such methods require a scoring function at the entity level, measuring whether each set of mentions is internally consistent. Coreference resolution can then be viewed as the following optimization,

$$\max_{\mathbf{z}} \quad \sum_{e=1} \psi_E(\{i : z_i = e\}), \tag{15.13}$$

where z_i indicates the entity referenced by mention i , and $\psi_E(\{i : z_i = e\})$ is a scoring function applied to all mentions i that are assigned to entity e .

Entity-based coreference resolution is conceptually similar to the unsupervised clustering problems encountered in chapter 5: the goal is to obtain clusters of mentions that are internally coherent. The number of possible clusterings is the **Bell number**, which is

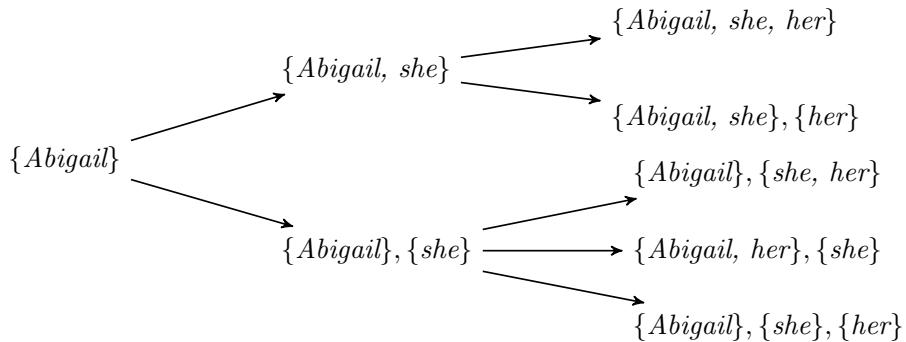


Figure 15.4: The Bell Tree for the sentence *Abigail hopes she speaks with her*. Which paths are excluded by the syntactic constraints mentioned in § 15.1.1?

defined by the following recurrence (Bell, 1934; Luo et al., 2004),

$$B_n = \sum_{k=0}^{n-1} B_k \binom{n-1}{k} = \frac{1}{e} \sum_{k=0}^{\infty} \frac{k^n}{k!}. \quad [15.14]$$

This recurrence is illustrated by the Bell tree, which is applied to a short coreference problem in Figure 15.4. The Bell number B_n grows exponentially with n , making exhaustive search of the space of clusterings impossible. For this reason, entity-based coreference resolution typically involves incremental search, in which clustering decisions are based on local evidence, in the hope of approximately optimizing the full objective in Equation 15.13. This approach is sometimes called **cluster ranking**, in contrast to mention ranking.

***Generative models of coreference** Entity-based coreference can be approached through probabilistic **generative models**, in which the mentions in the document are conditioned on a set of latent entities (Haghghi and Klein, 2007, 2010). An advantage of these methods is that they can be learned from unlabeled data (Poon and Domingos, 2008, e.g.); a disadvantage is that probabilistic inference is required not just for learning, but also for prediction. Furthermore, generative models require independence assumptions that are difficult to apply in coreference resolution, where the diverse and heterogeneous features do not admit an easy decomposition into mutually independent subsets.

15.2.4.1 Incremental cluster ranking

The SMASH method (§ 15.1.1) can be extended to entity-based coreference resolution by building up coreference clusters while moving through the document (Cardie and Wagstaff, 1999). At each mention, the algorithm iterates backwards through possible antecedent

7896 clusters; but unlike SMASH, a cluster is selected only if *all* members of its cluster are compatible
 7897 with the current mention. As mentions are added to a cluster, so are their features
 7898 (e.g., gender, number, animacy). In this way, incoherent chains like *{Hillary Clinton, Clinton, Bill Clinton}*
 7899 can be avoided. However, an incorrect assignment early in the document — a **search error**
 7900 — might lead to a cascade of errors later on.

7901 More sophisticated search strategies can help to ameliorate the risk of search errors.
 7902 One approach is **beam search** (§ 11.3), in which a set of hypotheses is maintained through-
 7903 out search. Each hypothesis represents a path through the Bell tree (Figure 15.4). Hy-
 7904 potheses are “expanded” either by adding the next mention to an existing cluster, or by
 7905 starting a new cluster. Each expansion receives a score, based on Equation 15.13, and the
 7906 top K hypotheses are kept on the beam as the algorithm moves to the next step.

7907 Incremental cluster ranking can be made more accurate by performing multiple passes
 7908 over the document, applying rules (or “sieves”) with increasing recall and decreasing
 7909 precision at each pass (Lee et al., 2013). In the early passes, coreference links are pro-
 7910 posed only between mentions that are highly likely to corefer (e.g., exact string match
 7911 for full names and nominals). Information can then be shared among these mentions,
 7912 so that when more permissive matching rules are applied later, agreement is preserved
 7913 across the entire cluster. For example, in the case of *{Hillary Clinton, Clinton, she}*, the
 7914 name-matching sieve would link *Clinton* and *Hillary Clinton*, and the pronoun-matching
 7915 sieve would then link *she* to the combined cluster. A deterministic multi-pass system
 7916 won nearly every track of the 2011 CoNLL shared task on coreference resolution (Prad-
 7917 han et al., 2011). Given the dominance of machine learning in virtually all other areas
 7918 of natural language processing — and more than fifteen years of prior work on machine
 7919 learning for coreference — this was a surprising result, even if learning-based methods
 7920 have subsequently regained the upper hand (e.g., Lee et al., 2017, the state-of-the-art at
 7921 the time of this writing).

7922 15.2.4.2 Incremental perceptron

Incremental coreference resolution can be learned with the **incremental perceptron**, as described in § 11.3.2. At mention i , each hypothesis on the beam corresponds to a clustering of mentions $1 \dots i - 1$, or equivalently, a path through the Bell tree up to position $i - 1$. As soon as none of the hypotheses on the beam are compatible with the gold coreference clustering, a perceptron update is made (Daumé III and Marcu, 2005). For concreteness, consider a linear cluster ranking model,

$$\psi_E(\{i : z_i = e\}) = \sum_{i:z_i=e} \theta \cdot f(i, \{j : j < i \wedge z_j = e\}), \quad [15.15]$$

7923 where the score for each cluster is computed as the sum of scores of all mentions that are
 7924 linked into the cluster, and $f(i, \emptyset)$ is a set of features for the non-anaphoric mention that
 7925 initiates the cluster.

7926 Using Figure 15.4 as an example, suppose that the ground truth is,

$$\mathbf{c}^* = \{\text{Abigail}, \text{her}\}, \{\text{she}\}, \quad [15.16]$$

7927 but that with a beam of size one, the learner reaches the hypothesis,

$$\hat{\mathbf{c}} = \{\text{Abigail}, \text{she}\}. \quad [15.17]$$

This hypothesis is incompatible with \mathbf{c}^* , so an update is needed:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{f}(\mathbf{c}^*) - \mathbf{f}(\hat{\mathbf{c}}) \quad [15.18]$$

$$= \boldsymbol{\theta} + (\mathbf{f}(\text{Abigail}, \emptyset) + \mathbf{f}(\text{she}, \emptyset)) - (\mathbf{f}(\text{Abigail}, \emptyset) + \mathbf{f}(\text{she}, \{\text{Abigail}\})) \quad [15.19]$$

$$= \boldsymbol{\theta} + \mathbf{f}(\text{she}, \emptyset) - \mathbf{f}(\text{she}, \{\text{Abigail}\}). \quad [15.20]$$

7928 This style of incremental update can also be applied to a margin loss between the gold
 7929 clustering and the top clustering on the beam. By backpropagating from this loss, it is also
 7930 possible to train a more complicated scoring function, such as a neural network in which
 7931 the score for each entity is a function of embeddings for the entity mentions (Wiseman
 7932 et al., 2015).

7933 15.2.4.3 Reinforcement learning

7934 **Reinforcement learning** is a topic worthy of a textbook of its own (Sutton and Barto,
 7935 1998),⁵ so this section will provide only a very brief overview, in the context of coreference
 7936 resolution. A stochastic **policy** assigns a probability to each possible **action**, conditional
 7937 on the context. The goal is to learn a policy that achieves a high expected reward, or
 7938 equivalently, a low expected cost.

7939 In incremental cluster ranking, a complete clustering on M mentions can be produced
 7940 by a sequence of M actions, in which the action z_i either merges mention i with an existing
 7941 cluster or begins a new cluster. We can therefore create a stochastic policy using the cluster
 7942 scores (Clark and Manning, 2016),

$$\Pr(z_i = e; \boldsymbol{\theta}) = \frac{\exp \psi_E(i \cup \{j : z_j = e\}; \boldsymbol{\theta})}{\sum_{e'} \exp \psi_E(i \cup \{j : z_j = e'\}; \boldsymbol{\theta})}, \quad [15.21]$$

7943 where $\psi_E(i \cup \{j : z_j = e\}; \boldsymbol{\theta})$ is the score under parameters $\boldsymbol{\theta}$ for assigning mention i to
 7944 cluster e . This score can be an arbitrary function of the mention i , the cluster e and its
 7945 (possibly empty) set of mentions; it can also include the history of actions taken thus far.

⁵A draft of the second edition can be found here: <http://incompleteideas.net/book/the-book-2nd.html>. Reinforcement learning has been used in spoken dialogue systems (Walker, 2000) and text-based game playing (Branavan et al., 2009), and was applied to coreference resolution by Clark and Manning (2015).

7946 If a policy assigns probability $p(c; \theta)$ to clustering c , then its expected loss is,

$$L(\theta) = \sum_{c \in \mathcal{C}(m)} p_\theta(c) \times \ell(c), \quad [15.22]$$

7947 where $\mathcal{C}(m)$ is the set of possible clusterings for mentions m . The loss $\ell(c)$ can be based on
 7948 any arbitrary scoring function, including the complex evaluation metrics used in corefer-
 7949 ence resolution (see § 15.4). This is an advantage of reinforcement learning, which can be
 7950 trained directly on the evaluation metric — unlike traditional supervised learning, which
 7951 requires a loss function that is differentiable and decomposable across individual deci-
 7952 sions.

Rather than summing over the exponentially many possible clusterings, we can approximate the expectation by sampling trajectories of actions, $z = (z_1, z_2, \dots, z_M)$, from the current policy. Each action z_i corresponds to a step in the Bell tree: adding mention m_i to an existing cluster, or forming a new cluster. Each trajectory z corresponds to a single clustering c , and so we can write the loss of an action sequence as $\ell(c(z))$. The **policy gradient** algorithm computes the gradient of the expected loss as an expectation over trajectories (Sutton et al., 2000),

$$\frac{\partial}{\partial \theta} L(\theta) = E_{z \sim \mathcal{Z}(m)} \ell(c(z)) \sum_{i=1}^M \frac{\partial}{\partial \theta} \log p(z_i | z_{1:i-1}, m) \quad [15.23]$$

$$\approx \frac{1}{K} \sum_{k=1}^K \ell(c(z^{(k)})) \sum_{i=1}^M \frac{\partial}{\partial \theta} \log p(z_i^{(k)} | z_{1:i-1}^{(k)}, m) \quad [15.24]$$

[15.25]

7953 where the action sequence $z^{(k)}$ is sampled from the current policy. Unlike the incremental
 7954 perceptron, an update is not made until the complete action sequence is available.

7955 15.2.4.4 Learning to search

7956 Policy gradient can suffer from high variance: while the average loss over K samples is
 7957 asymptotically equal to the expected reward of a given policy, this estimate may not be
 7958 accurate unless K is very large. This can make it difficult to allocate credit and blame to
 7959 individual actions. In **learning to search**, this problem is addressed through the addition
 7960 of an **oracle** policy, which is known to receive zero or small loss. The oracle policy can be
 7961 used in two ways:

- 7962 • The oracle can be used to generate partial hypotheses that are likely to score well,
 7963 by generating i actions from the initial state. These partial hypotheses are then used
 7964 as starting points for the learned policy. This is known as **roll-in**.

Algorithm 18 Learning to search for entity-based coreference resolution

```

1: procedure COMPUTE-GRADIENT(mentions  $m$ , loss function  $\ell$ , parameters  $\theta$ )
2:    $L(\theta) \leftarrow 0$ 
3:    $z \sim p(z | m; \theta)$                                  $\triangleright$  Sample a trajectory from the current policy
4:   for  $i \in \{1, 2, \dots, M\}$  do
5:     for action  $z \in \mathcal{Z}(z_{1:i-1}, m)$  do           $\triangleright$  All possible actions after history  $z_{1:i-1}$ 
6:        $h \leftarrow z_{1:i-1} \oplus z$                        $\triangleright$  Concatenate history  $z_{1:i-1}$  with action  $z$ 
7:       for  $j \in \{i+1, i+2, \dots, M\}$  do            $\triangleright$  Roll-out
8:          $h_j \leftarrow \operatorname{argmin}_h \ell(h_{1:j-1} \oplus h)$      $\triangleright$  Oracle selects action with minimum loss
9:        $L(\theta) \leftarrow L(\theta) + p(z | z_{1:i-1}, m; \theta) \times \ell(h)$        $\triangleright$  Update expected loss
10:      return  $\frac{\partial}{\partial \theta} L(\theta)$ 

```

- 7965 • The oracle can be used to compute the minimum possible loss from a given state, by
 7966 generating $M - i$ actions from the current state until completion. This is known as
 7967 **roll-out**.

7968 The oracle can be combined with the existing policy during both roll-in and roll-out, sam-
 7969 pling actions from each policy (Daumé III et al., 2009). One approach is to gradually
 7970 decrease the number of actions drawn from the oracle over the course of learning (Ross
 7971 et al., 2011).

7972 In the context of entity-based coreference resolution, Clark and Manning (2016) use
 7973 the learned policy for roll-in and the oracle policy for roll-out. Algorithm 18 shows how
 7974 the gradients on the policy weights are computed in this case. In this application, the
 7975 oracle is “noisy”, because it selects the action that minimizes only the *local* loss — the
 7976 accuracy of the coreference clustering up to mention i — rather than identifying the action
 7977 sequence that will lead to the best final coreference clustering on the entire document.
 7978 When learning from noisy oracles, it can be helpful to mix in actions from the current
 7979 policy with the oracle during roll-out (Chang et al., 2015).

7980 **15.3 Representations for coreference resolution**

7981 Historically, coreference resolution has employed an array of hand-engineered features
 7982 to capture the linguistic constraints and preferences described in § 15.1 (Soon et al., 2001).
 7983 Later work has documented the utility of lexical and bilexical features on mention pairs (Björkelund
 7984 and Nugues, 2011; Durrett and Klein, 2013). The most recent and successful methods re-
 7985 place many (but not all) of these features with distributed representations of mentions
 7986 and entities (Wiseman et al., 2015; Clark and Manning, 2016; Lee et al., 2017).

7987 **15.3.1 Features**

7988 Coreference features generally rely on a preprocessing pipeline to provide part-of-speech
 7989 tags and phrase structure parses. This pipeline makes it possible to design features that
 7990 capture many of the phenomena from § 15.1, and is also necessary for typical approaches
 7991 to mention identification. However, the pipeline may introduce errors that propagate
 7992 to the downstream coreference clustering system. Furthermore, the existence of such
 7993 a pipeline presupposes resources such as treebanks, which do not exist for many lan-
 7994 guages.⁶

7995 **15.3.1.1 Mention features**

7996 Features of individual mentions can help to predict anaphoricity. In systems where men-
 7997 tion detection is performed jointly with coreference resolution, these features can also
 7998 predict whether a span of text is likely to be a mention. For mention i , typical features
 7999 include:

8000 **Mention type.** Each span can be identified as a pronoun, name, or nominal, using the
 8001 part-of-speech of the head word of the mention: both the Penn Treebank and Uni-
 8002 versal Dependencies tagsets (§ 8.1.1) include tags for pronouns and proper nouns,
 8003 and all other heads can be marked as nominals (Haghghi and Klein, 2009).

8004 **Mention width.** The number of tokens in a mention is a rough predictor of its anaphor-
 8005 icity, with longer mentions being less likely to refer back to previously-defined enti-
 8006 ties.

8007 **Lexical features.** The first, last, and head words can help to predict anaphoricity; they are
 8008 also useful in conjunction with features such as mention type and part-of-speech,
 8009 providing a rough measure of agreement (Björkelund and Nugues, 2011). The num-
 8010 ber of lexical features can be very large, so it can be helpful to select only frequently-
 8011 occurring features (Durrett and Klein, 2013).

8012 **Morphosyntactic features.** These features include the part-of-speech, number, gender,
 8013 and dependency ancestors.

8014 The features for mention i and candidate antecedent a can be conjoined, producing
 8015 joint features that can help to assess the compatibility of the two mentions. For example,
 8016 Durrett and Klein (2013) conjoin each feature with the mention types of the anaphora
 8017 and the antecedent. Coreference resolution corpora such as ACE and OntoNotes contain

⁶The Universal Dependencies project has produced dependency treebanks for more than sixty languages. However, coreference features and mention detection are generally based on phrase structure trees, which exist for roughly two dozen languages. A list is available here: <https://en.wikipedia.org/wiki/Treebank>

8018 documents from various genres. By conjoining the genre with other features, it is possible
8019 to learn genre-specific feature weights.

8020 **15.3.1.2 Mention-pair features**

8021 For any pair of mentions i and j , typical features include:

8022 **Distance.** The number of intervening tokens, mentions, and sentences between i and j
8023 can all be used as distance features. These distances can be computed on the surface
8024 text, or on a transformed representation reflecting the breadth-first tree traversal
8025 (Figure 15.3). Rather than using the distances directly, they are typically binned,
8026 creating binary features.

8027 **String match.** A variety of string match features can be employed: exact match, suffix
8028 match, head match, and more complex matching rules that disregard irrelevant
8029 modifiers (Soon et al., 2001).

8030 **Compatibility.** Building on the model, features can measure the anaphor and antecedent
8031 agree with respect to morphosyntactic attributes such as gender, number, and ani-
8032 macy.

8033 **Nesting.** If one mention is nested inside another (e.g., *[The President of [France]]*), they
8034 generally cannot corefer.

8035 **Same speaker.** For documents with quotations, such as news articles, personal pronouns
8036 can be resolved only by determining the speaker for each mention (Lee et al., 2013).
8037 Coreference is also more likely between mentions from the same speaker.

8038 **Gazetteers.** These features indicate that the anaphor and candidate antecedent appear in
8039 a gazetteer of acronyms (e.g., *USA/United States*, *GATech/Georgia Tech*), demonymns
8040 (e.g., *Israel/Israeli*), or other aliases (e.g., *Knickerbockers/New York Knicks*).

8041 **Lexical semantics.** These features use a lexical resource such as WordNet to determine
8042 whether the head words of the mentions are related through synonymy, antonymy,
8043 and hypernymy (§ 4.2).

8044 **Dependency paths.** The dependency path between the anaphor and candidate antecedent
8045 can help to determine whether the pair can corefer, under the government and bind-
8046 ing constraints described in § 15.1.1.

8047 Comprehensive lists of mention-pair features are offered by Bengtson and Roth (2008) and
8048 Rahman and Ng (2011). Neural network approaches use far fewer mention-pair features:
8049 for example, Lee et al. (2017) include only speaker, genre, distance, and mention width
8050 features.

8051 **Semantics** In many cases, coreference seems to require knowledge and semantic in-
 8052 ferences, as in the running example, where we link *China* with a *country* and a *growth*
 8053 *market*. Some of this information can be gleaned from WordNet, which defines a graph
 8054 over **synsets** (see § 4.2). For example, one of the synsets of *China* is an instance of an
 8055 *Asian_nation#1*, which in turn is a hyponym of *country#2*, a synset that includes
 8056 *country*.⁷ Such paths can be used to measure the similarity between concepts (Pedersen
 8057 et al., 2004), and this similarity can be incorporated into coreference resolution as a fea-
 8058 ture (Ponzetto and Strube, 2006). Similar ideas can be applied to knowledge graphs in-
 8059 duced from Wikipedia (Ponzetto and Strube, 2007). But while such approaches improve
 8060 relatively simple classification-based systems, they have proven less useful when added
 8061 to the current generation of techniques.⁸ For example, Durrett and Klein (2013) employ
 8062 a range of semantics-based features — WordNet synonymy and hypernymy relations on
 8063 head words, named entity types (e.g., person, organization), and unsupervised clustering
 8064 over nominal heads — but find that these features give minimal improvement over a
 8065 baseline system using surface features.

8066 15.3.1.3 Entity features

8067 Many of the features for entity-mention coreference are generated by aggregating mention-
 8068 pair features over all mentions in the candidate entity (Culotta et al., 2007; Rahman and
 8069 Ng, 2011). Specifically, for each binary mention-pair feature $f(i, j)$, we compute the fol-
 8070 lowing entity-mention features for mention i and entity $e = \{j : j < i \wedge z_j = e\}$.

- 8071 • ALL-TRUE: Feature $f(i, j)$ holds for all mentions $j \in e$.
- 8072 • MOST-TRUE: Feature $f(i, j)$ holds for at least half and fewer than all mentions $j \in e$.
- 8073 • MOST-FALSE: Feature $f(i, j)$ holds for at least one and fewer than half of all men-
 8074 tions $j \in e$.
- 8075 • NONE: Feature $f(i, j)$ does not hold for any mention $j \in e$.

8076 For scalar mention-pair features (e.g., distance features), aggregation can be performed by
 8077 computing the minimum, maximum, and median values across all mentions in the cluster.
 8078 Additional entity-mention features include the number of mentions currently clustered in
 8079 the entity, and ALL-X and MOST-X features for each mention type.

8080 15.3.2 Distributed representations of mentions and entities

8081 Recent work has emphasized distributed representations of both mentions and entities.
 8082 One potential advantage is that pre-trained embeddings could help to capture the se-

⁷teletype font is used to indicate wordnet synsets, and *italics* is used to indicate strings.

⁸This point was made by Michael Strube at a 2015 workshop, noting that as the quality of the machine learning models in coreference has improved, the benefit of including semantics has become negligible.

8083 mantic compatibility underlying nominal coreference, helping with difficult cases like
 8084 (*Apple, the firm*) and (*China, the firm's biggest growth market*). Furthermore, a distributed
 8085 representation of entities can be trained to capture semantic features that are added by
 8086 each mention.

8087 **15.3.2.1 Mention embeddings**

8088 Entity mentions can be embedded into a vector space, providing the base layer for neural
 8089 networks that score coreference decisions (Wiseman et al., 2015).

8090 **Constructing the mention embedding** Various approaches for embedding multiword
 8091 units can be applied (see § 14.8). Figure 15.5 shows a recurrent neural network approach,
 8092 which begins by running a bidirectional LSTM over the entire text, obtaining hidden states
 8093 from the left-to-right and right-to-left passes, $\mathbf{h}_m = [\overleftarrow{\mathbf{h}}_m; \overrightarrow{\mathbf{h}}_m]$. Each candidate mention
 8094 span (s, t) is then represented by the vertical concatenation of four vectors:

$$\mathbf{u}^{(s,t)} = [\mathbf{u}_{\text{first}}^{(s,t)}; \mathbf{u}_{\text{last}}^{(s,t)}; \mathbf{u}_{\text{head}}^{(s,t)}; \phi^{(s,t)}], \quad [15.26]$$

8095 where $\mathbf{u}_{\text{first}}^{(s,t)} = \mathbf{h}_{s+1}$ is the embedding of the first word in the span, $\mathbf{u}_{\text{last}}^{(s,t)} = \mathbf{h}_t$ is the
 8096 embedding of the last word, $\mathbf{u}_{\text{head}}^{(s,t)}$ is the embedding of the “head” word, and $\phi^{(s,t)}$ is a
 8097 vector of surface features, such as the length of the span (Lee et al., 2017).

Attention over head words Rather than identifying the head word from the output of a parser, it can be computed from a neural **attention mechanism**:

$$\tilde{\alpha}_m = \theta_\alpha \cdot \mathbf{h}_m \quad [15.27]$$

$$\mathbf{a}^{(s,t)} = \text{SoftMax}([\tilde{\alpha}_{s+1}, \tilde{\alpha}_{s+2}, \dots, \tilde{\alpha}_t]) \quad [15.28]$$

$$\mathbf{u}_{\text{head}}^{(s,t)} = \sum_{m=s+1}^t a_m^{(s,t)} \mathbf{h}_m. \quad [15.29]$$

8098 Each token m gets a scalar score $\tilde{\alpha}_m = \theta_\alpha \cdot \mathbf{h}_m$, which is the dot product of the LSTM
 8099 hidden state \mathbf{h}_m and a vector of weights θ_α . The vector of scores for tokens in the span
 8100 $m \in \{s + 1, s + 2, \dots, t\}$ is then passed through a softmax layer, yielding a vector $\mathbf{a}^{(s,t)}$
 8101 that allocates one unit of attention across the span. This eliminates the need for syntactic
 8102 parsing to recover the head word; instead, the model learns to identify the most important
 8103 words in each span. Attention mechanisms were introduced in neural machine transla-
 8104 tion (Bahdanau et al., 2014), and are described in more detail in § 18.3.1.

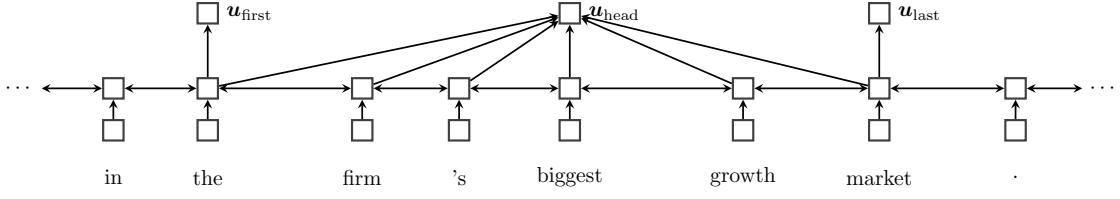


Figure 15.5: A bidirectional recurrent model of mention embeddings. The mention is represented by its first word, its last word, and an estimate of its head word, which is computed from a weighted average (Lee et al., 2017).

Using mention embeddings Given a set of mention embeddings, each mention i and candidate antecedent a is scored as,

$$\psi(a, i) = \psi_S(a) + \psi_S(i) + \psi_M(a, i) \quad [15.30]$$

$$\psi_S(a) = \text{FeedForward}_S(\mathbf{u}^{(a)}) \quad [15.31]$$

$$\psi_S(i) = \text{FeedForward}_S(\mathbf{u}^{(i)}) \quad [15.32]$$

$$\psi_M(a, i) = \text{FeedForward}_M([\mathbf{u}^{(a)}; \mathbf{u}^{(i)}; \mathbf{u}^{(a)} \odot \mathbf{u}^{(i)}; \mathbf{f}(a, i, \mathbf{w})]), \quad [15.33]$$

where $\mathbf{u}^{(a)}$ and $\mathbf{u}^{(i)}$ are the embeddings for spans a and i respectively, as defined in Equation 15.26.

- The scores $\psi_S(a)$ quantify whether span a is likely to be a coreferring mention, independent of what it corefers with. This allows the model to learn identify mentions directly, rather than identifying mentions with a preprocessing step.
- The score $\psi_M(a, i)$ computes the compatibility of spans a and i . Its base layer is a vector that includes the embeddings of spans a and i , their elementwise product $\mathbf{u}^{(a)} \odot \mathbf{u}^{(i)}$, and a vector of surface features $\mathbf{f}(a, i, \mathbf{w})$, including distance, speaker, and genre information.

Lee et al. (2017) provide an error analysis that shows how this method can correctly link a *blaze* and a *fire*, while incorrectly linking *pilots* and *fight attendants*. In each case, the coreference decision is based on similarities in the word embeddings.

Rather than embedding individual mentions, Clark and Manning (2016) embed mention pairs. At the base layer, their network takes embeddings of the words in and around each mention, as well as **one-hot** vectors representing a few surface features, such as the distance and string matching features. This base layer is then passed through a multilayer feedforward network with ReLU nonlinearities, resulting in a representation of the mention pair. The output of the mention pair encoder $\mathbf{u}_{i,j}$ is used in the scoring function of a mention-ranking model, $\psi_M(i, j) = \theta \cdot \mathbf{u}_{i,j}$. A similar approach is used to score cluster

8124 pairs, constructing a cluster-pair encoding by **pooling** over the mention-pair encodings
8125 for all pairs of mentions within the two clusters.

8126 **15.3.2.2 Entity embeddings**

8127 In entity-based coreference resolution, each entity should be represented by properties of
8128 its mentions. In a distributed setting, we maintain a set of vector entity embeddings, v_e .
8129 Each candidate mention receives an embedding u_i ; Wiseman et al. (2016) compute this
8130 embedding by a single-layer neural network, applied to a vector of surface features. The
8131 decision of whether to merge mention i with entity e can then be driven by a feedforward
8132 network, $\psi_E(i, e) = \text{Feedforward}([v_e; u_i])$. If i is added to entity e , then its representa-
8133 tion is updated recurrently, $v_e \leftarrow f(v_e, u_i)$, using a recurrent neural network such as a
8134 long short-term memory (LSTM; chapter 6). Alternatively, we can apply a **pooling** oper-
8135 ation, such as max-pooling or average-pooling (chapter 3), setting $v_e \leftarrow \text{Pool}(v_e, u_i)$. In
8136 either case, the update to the representation of entity e can be thought of as adding new
8137 information about the entity from mention i .

8138 **15.4 Evaluating coreference resolution**

8139 The state of coreference evaluation is aggravatingly complex. Early attempts at sim-
8140 ple evaluation metrics were found to under-penalize trivial baselines, such as placing
8141 each mention in its own cluster, or grouping all mentions into a single cluster. Follow-
8142 ing Denis and Baldridge (2009), the CoNLL 2011 shared task on coreference (Pradhan
8143 et al., 2011) formalized the practice of averaging across three different metrics: MUC (Vi-
8144 lain et al., 1995), B-CUBED (Bagga and Baldwin, 1998a), and CEAf (Luo, 2005). Refer-
8145 ence implementations of these metrics are available from Pradhan et al. (2014) at <https://github.com/conll/reference-coreference-scorers>.
8146

8147 **Additional resources**

8148 Ng (2010) surveys coreference resolution through 2010. Early work focused exclusively
8149 on pronoun resolution, with rule-based (Lappin and Leass, 1994) and probabilistic meth-
8150 ods (Ge et al., 1998). The full coreference resolution problem was popularized in a shared
8151 task associated with the sixth Message Understanding Conference, which included coref-
8152 erence annotations for training and test sets of thirty documents each (Grishman and
8153 Sundheim, 1996). An influential early paper was the decision tree approach of Soon et al.
8154 (2001), who introduced mention ranking. A comprehensive list of surface features for
8155 coreference resolution is offered by Bengtson and Roth (2008). Durrett and Klein (2013)
8156 improved on prior work by introducing a large lexicalized feature set; subsequent work
8157 has emphasized neural representations of entities and mentions (Wiseman et al., 2015).

8158 **Exercises**

8159 1. Select an article from today's news, and annotate coreference for the first twenty
 8160 noun phrases that appear in the article (include nested noun phrases). That is,
 8161 group the noun phrases into entities, where each entity corresponds to a set of noun
 8162 phrases. Then specify the mention-pair training data that would result from the first
 8163 five noun phrases.

8164 2. Using your annotations from the preceding problem, compute the following statistics:
 8165

- 8166 • The number of times new entities are introduced by each of the three types of
 8167 referring expressions: pronouns, proper nouns, and nominals. Include "singleton"
 8168 entities that are mentioned only once.
- 8169 • For each type of referring expression, compute the fraction of mentions that are
 8170 anaphoric.

8171 3. Apply a simple heuristic to all pronouns in the article from the previous exercise.
 8172 Specifically, link each pronoun to the closest preceding noun phrase that agrees in
 8173 gender, number, animacy, and person. Compute the following evaluation:

- 8174 • True positive: a pronoun that is linked to a noun phrase with which it is coref-
 8175 erent, or is correctly labeled as the first mention of an entity.
- 8176 • False positive: a pronoun that is linked to a noun phrase with which it is not
 8177 coreferent. (This includes mistakenly linking singleton or non-referential pro-
 8178 nouns.)
- 8179 • False negative: a pronoun that is not linked to a noun phrase with which it is
 8180 coreferent.

8181 Compute the *F-MEASURE* for your method, and for a trivial baseline in which ev-
 8182 ery mention is its own entity. Are there any additional heuristics that would have
 8183 improved the performance of this method?

8184 4. Durrett and Klein (2013) compute the probability of the gold coreference clustering
 8185 by summing over all antecedent structures that are compatible with the clustering.
 8186 Compute the number of antecedent structures for a single entity with K mentions.

8187 5. Use the policy gradient algorithm to compute the gradient for the following sce-
 8188 nario, based on the Bell tree in Figure 15.4:

- 8189 • The gold clustering c^* is $\{Abigail, her\}, \{she\}$.

- Drawing a single sequence of actions ($K = 1$) from the current policy, you obtain the following incremental clusterings:

$$\begin{aligned}\mathbf{c}(a_1) &= \{\text{Abigail}\} \\ \mathbf{c}(\mathbf{a}_{1:2}) &= \{\text{Abigail}, \text{she}\} \\ \mathbf{c}(\mathbf{a}_{1:3}) &= \{\text{Abigail}, \text{she}\}, \{\text{her}\}.\end{aligned}$$

- 8190 • At each mention t , the action space \mathcal{A}_t is to merge the mention with each existing cluster, or the empty cluster, with probability,
- 8191

$$\Pr(\text{Merge}(m_t, \mathbf{c}(\mathbf{a}_{1:t-1}))) \propto \exp \psi_E(m_t \cup \mathbf{c}(\mathbf{a}_{1:t-1})), \quad [15.34]$$

8192 where the cluster score $\psi_E(m_t \cup c)$ is defined in Equation 15.15.

8193 Compute the gradient $\frac{\partial}{\partial \theta} L(\theta)$ in terms of the loss $\ell(c(a))$ and the features of each
 8194 (potential) cluster. Explain the differences between the gradient-based update $\theta \leftarrow \theta - \frac{\partial}{\partial \theta} L(\theta)$
 8195 and the incremental perceptron update from this sample example.

8196 **Chapter 16**

8197 **Discourse**

8198 Applications of natural language processing often concern multi-sentence documents:
8199 from paragraph-long restaurant reviews, to 500-word newspaper articles, to 500-page
8200 novels. Yet most of the methods that we have discussed thus far are concerned with
8201 individual sentences. This chapter discusses theories and methods for handling multi-
8202 sentence linguistic phenomena, known collectively as **discourse**. There are diverse char-
8203 acterizations of discourse structure, and no single structure is ideal for every computa-
8204 tional application. This chapter covers some of the most well studied discourse repre-
8205 sentations, while highlighting computational models for identifying and exploiting these
8206 structures.

8207 **16.1 Segments**

8208 A document or conversation can be viewed as a sequence of **segments**, each of which is
8209 **cohesive** in its content and/or function. In Wikipedia biographies, these segments often
8210 pertain to various aspects to the subject's life: early years, major events, impact on others,
8211 and so on. This segmentation is organized around **topics**. Alternatively, scientific research
8212 articles are often organized by **functional themes**: the introduction, a survey of previous
8213 research, experimental setup, and results.

8214 Written texts often mark segments with section headers and related formatting de-
8215 vices. However, such formatting may be too coarse-grained to support applications such
8216 as the retrieval of specific passages of text that are relevant to a query (Hearst, 1997).
8217 Unformatted speech transcripts, such as meetings and lectures, are also an application
8218 scenario for segmentation (Carletta, 2007; Glass et al., 2007; Janin et al., 2003).

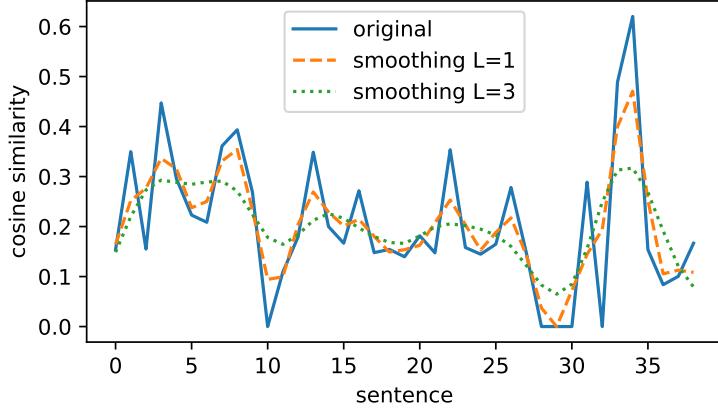


Figure 16.1: Smoothed cosine similarity among adjacent sentences in a news article. Local minima at $m = 10$ and $m = 29$ indicate likely segmentation points.

8219 16.1.1 Topic segmentation

A cohesive topic segment forms a unified whole, using various linguistic devices: repeated references to an entity or event; the use of conjunctions to link related ideas; and the repetition of meaning through lexical choices (Halliday and Hasan, 1976). Each of these cohesive devices can be measured, and then used as features for topic segmentation. A classical example is the use of lexical cohesion in the `TextTiling` method for topic segmentation (Hearst, 1997). The basic idea is to compute the textual similarity between each pair of adjacent blocks of text (sentences or fixed-length units), using a formula such as the smoothed **cosine similarity** of their bag-of-words vectors,

$$s_m = \frac{\mathbf{x}_m \cdot \mathbf{x}_{m+1}}{\|\mathbf{x}_m\|_2 \times \|\mathbf{x}_{m+1}\|_2} \quad [16.1]$$

$$\bar{s}_m = \sum_{\ell=0}^L k_\ell (s_{m+\ell} + s_{m-\ell}), \quad [16.2]$$

8220 with k_ℓ representing the value of a smoothing kernel of size L , e.g. $\mathbf{k} = [1, 0.5, 0.25]^\top$.
 8221 Segmentation points are then identified at local minima in the smoothed similarities \bar{s} ,
 8222 since these points indicate changes in the overall distribution of words in the text. An
 8223 example is shown in Figure 16.1.

8224 Text segmentation can also be formulated as a probabilistic model, in which each seg-
 8225 ment has a unique language model that defines the probability over the text in the seg-
 8226 ment (Utiyama and Isahara, 2001; Eisenstein and Barzilay, 2008; Du et al., 2013).¹ A good

¹There is a rich literature on how latent variable models (such as **latent Dirichlet allocation**) can track

8227 segmentation achieves high likelihood by grouping segments with similar word distribu-
8228 tions. This probabilistic approach can be extended to **hierarchical topic segmentation**, in
8229 which each topic segment is divided into subsegments (Eisenstein, 2009). All of these ap-
8230 proaches are unsupervised. While labeled data can be obtained from well-formatted texts
8231 such as textbooks, such annotations may not generalize to speech transcripts in alterna-
8232 tive domains. Supervised methods have been tried in cases where in-domain labeled data
8233 is available, substantially improving performance by learning weights on multiple types
8234 of features (Galley et al., 2003).

8235 16.1.2 Functional segmentation

8236 In some genres, there is a canonical set of communicative *functions*: for example, in sci-
8237 entific research articles, one such function is to communicate the general background for
8238 the article, another is to introduce a new contribution, or to describe the aim of the re-
8239 search (Teufel et al., 1999). A **functional segmentation** divides the document into con-
8240 tiguous segments, sometimes called **rhetorical zones**, in which each sentence has the same
8241 function. Teufel and Moens (2002) train a supervised classifier to identify the functional
8242 of each sentence in a set of scientific research articles, using features that describe the sen-
8243 tence's position in the text, its similarity to the rest of the article and title, tense and voice of
8244 the main verb, and the functional role of the previous sentence. Functional segmentation
8245 can also be performed without supervision. Noting that some types of Wikipedia arti-
8246 cles have very consistent functional segmentations (e.g., articles about cities or chemical
8247 elements), Chen et al. (2009) introduce an unsupervised model for functional segmenta-
8248 tion, which learns both the language model associated with each function and the typical
8249 patterning of functional segments across the article.

8250 16.2 Entities and reference

8251 Another dimension of discourse relates to which entities are mentioned throughout the
8252 text, and how. Consider the examples in Figure 16.2: Grosz et al. (1995) argue that the first
8253 discourse is more coherent. Do you agree? The examples differ in their choice of **refe-
8254 ring expressions** for the protagonist *John*, and in the syntactic constructions in sentences
8255 (b) and (d). The examples demonstrate the need for theoretical models to explain how
8256 referring expressions are chosen, and where they are placed within sentences. Such mod-
8257 els can then be used to help interpret the overall structure of the discourse, to measure
8258 discourse coherence, and to generate discourses in which referring expressions are used
8259 coherently.

topics across documents (Blei et al., 2003; Blei, 2012).

- | | |
|--|---|
| (16.1) a. John went to his favorite music store to buy a piano.
b. He had frequented the store for many years.
c. He was excited that he could finally buy a piano.
d. He arrived just as the store was closing for the day | (16.2) a. John went to his favorite music store to buy a piano.
b. It was a store John had frequented for many years.
c. He was excited that he could finally buy a piano.
d. It was closing just as John arrived. |
|--|---|

Figure 16.2: Two tellings of the same story (Grosz et al., 1995). The discourse on the left uses referring expressions coherently, while the one on the right does not.

16.2.1 Centering theory

The relationship between discourse and entity reference is most elaborated in **centering theory** (Grosz et al., 1995). According to the theory, every utterance in the discourse is characterized by a set of entities, known as *centers*.

- The **forward-looking centers** in utterance m are all the entities that are mentioned in the utterance, $c_f(w_m) = \{e_1, e_2, \dots\}$. The forward-looking centers are partially ordered by their syntactic prominence, favoring subjects over other positions.
- The **backward-looking center** $c_b(w_m)$ is the highest-ranked element in the set of forward-looking centers from the previous utterance $c_f(w_{m-1})$ that is also mentioned in w_m .

Given these two definitions, centering theory makes the following predictions about the form and position of referring expressions:

1. If a pronoun appears in the utterance w_m , then the backward-looking center $c_b(w_m)$ must also be realized as a pronoun. This rule argues against the use of *it* to refer to the piano store in Example (16.2d), since JOHN is the backward looking center of (16.2d), and he is mentioned by name and not by a pronoun.
2. Sequences of utterances should retain the same backward-looking center if possible, and ideally, the backward-looking center should also be the top-ranked element in the list of forward-looking centers. This rule argues in favor of the preservation of JOHN as the backward-looking center throughout Example (16.1).

Centering theory unifies aspects of syntax, discourse, and anaphora resolution. However, it can be difficult to clarify exactly how to rank the elements of each utterance, or even how to partition a text or dialog into utterances (Poesio et al., 2004).

	SKYLER	WALTER	DANGER	A GUY	THE DOOR
<i>You don't know who you're talking to,</i>	S	-	-	-	-
<i>so let me clue you in.</i>	O	O	-	-	-
<i>I am not in danger, Skyler.</i>	X	S	X	-	-
<i>I am the danger.</i>	-	S	O	-	-
<i>A guy opens his door and gets shot,</i>	-	-	-	S	O
<i>and you think that of me?</i>	S	X	-	-	-
<i>No. I am the one who knocks!</i>	-	S	-	-	-

Figure 16.3: The entity grid representation for a dialogue from the television show *Breaking Bad*.

16.2.2 The entity grid

One way to formalize the ideas of centering theory is to arrange the entities in a text or conversation in an **entity grid**. This is a data structure with one row per sentence, and one column per entity (Barzilay and Lapata, 2008). Each cell $c(m, i)$ can take the following values:

$$c(m, i) = \begin{cases} S, & \text{entity } i \text{ is in subject position in sentence } m \\ O, & \text{entity } i \text{ is in object position in sentence } m \\ X, & \text{entity } i \text{ appears in sentence } m, \text{ in neither subject nor object position} \\ -, & \text{entity } i \text{ does not appear in sentence } m. \end{cases} \quad [16.3]$$

To populate the entity grid, syntactic parsing is applied to identify subject and object positions, and coreference resolution is applied to link multiple mentions of a single entity. An example is shown in Figure 16.3.

After the grid is constructed, the coherence of a document can be measured by the transitions between adjacent cells in each column. For example, the transition $(S \rightarrow S)$ keeps an entity in subject position across adjacent sentences; the transition $(O \rightarrow S)$ promotes an entity from object position to subject position; the transition $(S \rightarrow -)$ drops the subject of one sentence from the next sentence. The probabilities of each transition can be estimated from labeled data, and an entity grid can then be scored by the sum of the log-probabilities across all columns and all transitions, $\sum_{i=1}^{N_e} \sum_{m=1}^M \log p(c(m, i) | c(m-1, i))$. The resulting probability can be used as a proxy for the coherence of a text. This has been shown to be useful for a range of tasks: determining which of a pair of articles is more readable (Schwartz and Ostendorf, 2005), correctly ordering the sentences in a scrambled

8301 text (Lapata, 2003), and disentangling multiple conversational threads in an online multi-
 8302 party chat (Elsner and Charniak, 2010).

8303 **16.2.3 *Formal semantics beyond the sentence level**

8304 An alternative view of the role of entities in discourse focuses on formal semantics, and the
 8305 construction of meaning representations for multi-sentence units. Consider the following
 8306 two sentences (from Bird et al., 2009):

- 8307 (16.3) a. Angus owns a dog.
 8308 b. It bit Irene.

8309 We would like to recover the formal semantic representation,

$$\exists x. \text{DOG}(x) \wedge \text{OWN}(\text{ANGUS}, x) \wedge \text{BITE}(x, \text{IRENE}). \quad [16.4]$$

However, the semantic representations of each individual sentence are:

$$\exists x. \text{DOG}(x) \wedge \text{OWN}(\text{ANGUS}, x) \quad [16.5]$$

$$\text{BITE}(y, \text{IRENE}). \quad [16.6]$$

8310 Unifying these two representations into the form of Equation 16.4 requires linking the
 8311 unbound variable y from [16.6] with the quantified variable x in [16.5]. Discourse under-
 8312 standing therefore requires the reader to update a set of assignments, from variables
 8313 to entities. This update would (presumably) link the *dog* in the first sentence of [16.3]
 8314 with the unbound variable y in the second sentence, thereby licensing the conjunction in
 8315 [16.4].² This basic idea is at the root of **dynamic semantics** (Groenendijk and Stokhof,
 8316 1991). **Segmented discourse representation theory** links dynamic semantics with a set
 8317 of **discourse relations**, which explain how adjacent units of text are rhetorically or con-
 8318 ceptually related (Lascarides and Asher, 2007). The next section explores the theory of
 8319 discourse relations in more detail.

8320 **16.3 Relations**

8321 In dependency grammar, sentences are characterized by a graph (usually a tree) of syntac-
 8322 tic relations between words, such as NSUBJ and DET. A similar idea can be applied at the
 8323 document level, identifying relations between discourse units, such as clauses, sentences,
 8324 or paragraphs. The task of **discourse parsing** involves identifying discourse units and
 8325 the relations that hold between them. These relations can then be applied to tasks such as
 8326 document classification and summarization, as discussed in § 16.3.4.

²This linking task is similar to coreference resolution (see chapter 15), but here the connections are between semantic variables, rather than spans of text.

- TEMPORAL
 - Asynchronous
 - Synchronous: precedence, succession
- CONTINGENCY
 - Cause: result, reason
 - Pragmatic cause: justification
 - Condition: hypothetical, general, unreal present, unreal past, real present, real past
 - Pragmatic condition: relevance, implicit assertion
- COMPARISON
 - Contrast: juxtaposition, opposition
 - Pragmatic contrast
 - Concession: expectation, contra-expectation
 - Pragmatic concession
- EXPANSION
 - Conjunction
 - Instantiation
 - Restatement: specification, equivalence, generalization
 - Alternative: conjunctive, disjunctive, chosen alternative
 - Exception
 - List

Table 16.1: The hierarchy of discourse relation in the Penn Discourse Treebank annotations (Prasad et al., 2008). For example, PRECEDENCE is a subtype of SYNCHRONOUS, which is a type of TEMPORAL relation.

8327 16.3.1 Shallow discourse relations

8328 The existence of discourse relations is hinted by **discourse connectives**, such as *however*,
 8329 *moreover*, *meanwhile*, and *if ... then*. These connectives explicitly specify the relationship
 8330 between adjacent units of text: *however* signals a contrastive relationship, *moreover* signals
 8331 that the subsequent text elaborates or strengthens the point that was made immediately
 8332 beforehand, *meanwhile* indicates that two events are contemporaneous, and *if ... then* sets
 8333 up a conditional relationship. Discourse connectives can therefore be viewed as a starting
 8334 point for the analysis of discourse relations.

8335 In **lexicalized tree-adjoining grammar for discourse (D-LTAG)**, each connective an-
 8336 chors a relationship between two units of text (Webber, 2004). This model provides the
 8337 theoretical basis for the **Penn Discourse Treebank (PDTB)**, the largest corpus of discourse
 8338 relations in English (Prasad et al., 2008). It includes a hierarchical inventory of discourse
 8339 relations (shown in Table 16.1), which is created by abstracting the meanings implied by
 8340 the discourse connectives that appear in real texts (Knott, 1996). These relations are then
 8341 annotated on the same corpus of news text used in the Penn Treebank (see § 9.2.2), adding
 8342 the following information:

- 8343 • Each connective is annotated for the discourse relation or relations that it expresses,
 8344 if any — many discourse connectives have senses in which they do not signal a
 8345 discourse relation (Pitler and Nenkova, 2009).
- 8346 • For each discourse relation, the two arguments of the relation are specified as ARG1
 8347 and ARG2, where ARG2 is constrained to be adjacent to the connective. These argu-
 8348 ments may be sentences, but they may also smaller or larger units of text.
- 8349 • Adjacent sentences are annotated for **implicit discourse relations**, which are not
 8350 marked by any connective. When a connective could be inserted between a pair
 8351 of sentence, the annotator supplies it, and also labels its sense (e.g., example 16.5).
 8352 In some cases, there is no relationship at all between a pair of adjacent sentences;
 8353 in other cases, the only relation is that the adjacent sentences mention one or more
 8354 shared entity. These phenomena are annotated as NOREL and ENTRREL (entity rela-
 8355 tion), respectively.

8356 Examples of Penn Discourse Treebank annotations are shown in (16.4). In (16.4), the
 8357 word *therefore* acts as an explicit discourse connective, linking the two adjacent units of
 8358 text. The Treebank annotations also specify the “sense” of each relation, linking the con-
 8359 nective to a relation in the sense inventory shown in Table 16.1: in (16.4), the relation is
 8360 PRAGMATIC CAUSE:JUSTIFICATION because it relates to the author’s communicative in-
 8361 tentions. The word *therefore* can also signal causes in the external world (e.g., *He was*
 8362 *therefore forced to relinquish his plan*). In **discourse sense classification**, the goal is to de-
 8363 termine which discourse relation, if any, is expressed by each connective. A related task
 8364 is the classification of implicit discourse relations, as in (16.5). In this example, the re-
 8365 lationship between the adjacent sentences could be expressed by the connective *because*,
 8366 indicating a CAUSE:REASON relationship.

8367 16.3.1.1 Classifying explicit discourse relations and their arguments

8368 As suggested by the examples above, many connectives can be used to invoke multiple
 8369 types of discourse relations. Similarly, some connectives have senses that are unrelated
 8370 to discourse: for example, *and* functions as a discourse connective when it links propo-
 8371 sitions, but not when it links noun phrases (Lin et al., 2014). Nonetheless, the senses of
 8372 explicitly-marked discourse relations in the Penn Treebank are relatively easy to classify,
 8373 at least at the coarse-grained level. When classifying the four top-level PDTB relations,
 8374 90% accuracy can be obtained simply by selecting the most common relation for each
 8375 connective (Pitler and Nenkova, 2009). At the more fine-grained levels of the discourse
 8376 relation hierarchy, connectives are more ambiguous. This fact is reflected both in the ac-
 8377 curacy of automatic sense classification (Versley, 2011) and in interannotator agreement,
 8378 which falls to 80% for level-3 discourse relations (Prasad et al., 2008).

- (16.4) *...as this business of whaling has somehow come to be regarded among landsmen as a rather unpoetical and disreputable pursuit; therefore, I am all anxiety to convince ye, ye landsmen, of the injustice hereby done to us hunters of whales.*
- (16.5) But a few funds have taken other defensive steps. *Some have raised their cash positions to record levels. Implicit = BECAUSE High cash positions help buffer a fund when the market falls.*
- (16.6) Michelle lives in a hotel room, and although **she drives a canary-colored Porsche**, *she hasn't time to clean or repair it.*
- (16.7) *Most oil companies, when they set exploration and production budgets for this year, forecast revenue of \$15 for each barrel of crude produced.*

Figure 16.4: Example annotations of discourse relations. In the style of the Penn Discourse Treebank, the discourse connective is underlined, the first argument is shown in italics, and the second argument is shown in bold. Examples (16.5-16.7) are quoted from Prasad et al. (2008).

8379 A more challenging task for explicitly-marked discourse relations is to identify the
 8380 scope of the arguments. Discourse connectives need not be adjacent to ARG1, as shown
 8381 in item 16.6, where ARG1 follows ARG2; furthermore, the arguments need not be contigu-
 8382 ous, as shown in (16.7). For these reasons, recovering the arguments of each discourse
 8383 connective is a challenging subtask. Because intra-sentential arguments are often syn-
 8384 tactic constituents (see chapter 10), many approaches train a classifier to predict whether
 8385 each constituent is an appropriate argument for each explicit discourse connective (Well-
 8386 ner and Pustejovsky, 2007; Lin et al., 2014, e.g.).

8387 16.3.1.2 Classifying implicit discourse relations

Implicit discourse relations are considerably more difficult to classify and to annotate.³ Most approaches are based on an encoding of each argument, which is then used as input to a non-linear classifier:

$$\mathbf{z}^{(i)} = \text{Encode}(\mathbf{w}^{(i)}) \quad [16.7]$$

$$\mathbf{z}^{(i+1)} = \text{Encode}(\mathbf{w}^{(i+1)}) \quad [16.8]$$

$$\hat{y}_i = \underset{y}{\operatorname{argmax}} \Psi(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}). \quad [16.9]$$

³In the dataset for the 2015 shared task on shallow discourse parsing, the interannotator agreement was 91% for explicit discourse relations and 81% for non-explicit relations, across all levels of detail (Xue et al., 2015).

8388 This basic framework can be instantiated in several ways, including both feature-based
 8389 and neural encoders. Several recent approaches are compared in the 2015 and 2016 shared
 8390 tasks at the Conference on Natural Language Learning (Xue et al., 2015, 2016).

8391 **Feature-based approaches** Each argument can be encoded into a vector of surface fea-
 8392 tures. The encoding typically includes lexical features (all words, or all content words, or
 8393 a subset of words such as the first three and the main verb), Brown clusters of individ-
 8394 ual words (§ 14.4), and syntactic features such as terminal productions and dependency
 8395 arcs (Pitler et al., 2009; Lin et al., 2009; Rutherford and Xue, 2014). The classification func-
 8396 tion then has two parts. First, it creates a joint feature vector by combining the encodings
 8397 of each argument, typically by computing the cross-product of all features in each encod-
 8398 ing:

$$\mathbf{f}(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}) = \{(a \times b \times y) : (\mathbf{z}_a^{(i)} \mathbf{z}_b^{(i+1)})\} \quad [16.10]$$

8399 The size of this feature set grows with the square of the size of the vocabulary, so it can be
 8400 helpful to select a subset of features that are especially useful on the training data (Park
 8401 and Cardie, 2012). After \mathbf{f} is computed, any classifier can be trained to compute the final
 8402 score, $\Psi(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}) = \boldsymbol{\theta} \cdot \mathbf{f}(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)})$.

8403 **Neural network approaches** In neural network architectures, the encoder is learned
 8404 jointly with the classifier as an end-to-end model. Each argument can be encoded using
 8405 a variety of neural architectures (surveyed in § 14.8): recursive (§ 10.6.1; Ji and Eisenstein,
 8406 2015), recurrent (§ 6.3; Ji et al., 2016), and convolutional (§ 3.4; Qin et al., 2017). The clas-
 8407 sification function can then be implemented as a feedforward neural network on the two
 8408 encodings (chapter 3; for examples, see Rutherford et al., 2017; Qin et al., 2017), or as a
 8409 simple bilinear product, $\Psi(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}) = (\mathbf{z}^{(i)})^\top \boldsymbol{\Theta}_y \mathbf{z}^{(i+1)}$ (Ji and Eisenstein, 2015). The
 8410 encoding model can be trained by backpropagation from the classification objective, such
 8411 as the margin loss. Rutherford et al. (2017) show that neural architectures outperform
 8412 feature-based approaches in most settings. While neural approaches require engineering
 8413 the network architecture (e.g., embedding size, number of hidden units in the classifier),
 8414 feature-based approaches also require significant engineering to incorporate linguistic re-
 8415 sources such as Brown clusters and parse trees, and to select a subset of relevant features.

8416 16.3.2 Hierarchical discourse relations

8417 In sentence parsing, adjacent phrases combine into larger constituents, ultimately pro-
 8418 ducing a single constituent for the entire sentence. The resulting tree structure enables
 8419 structured analysis of the sentence, with subtrees that represent syntactically coherent
 8420 chunks of meaning. **Rhetorical Structure Theory (RST)** extends this style of hierarchical
 8421 analysis to the discourse level (Mann and Thompson, 1988).

8422 The basic element of RST is the **discourse unit**, which refers to a contiguous span of
 8423 text. **Elementary discourse units** (EDUs) are the atomic elements in this framework, and
 8424 are typically (but not always) clauses.⁴ Each discourse relation combines two or more
 8425 adjacent discourse units into a larger, composite discourse unit; this process ultimately
 8426 unites the entire text into a tree-like structure.⁵

8427 **Nuclearity** In many discourse relations, one argument is primary. For example:

8428 (16.8) [LaShawn loves animals]_N
 8429 [She has nine dogs and one pig]_S

8430 In this example, the second sentence provides EVIDENCE for the point made in the first
 8431 sentence. The first sentence is thus the **nucleus** of the discourse relation, and the second
 8432 sentence is the **satellite**. The notion of **nuclearity** is analogous to the head-modifier struc-
 8433 ture of dependency parsing (see § 11.1.1). However, in RST, some relations have multiple
 8434 nuclei. For example, the arguments of the CONTRAST relation are equally important:

8435 (16.9) [The clash of ideologies survives this treatment]_N
 8436 [but the nuance and richness of Gorky's individual characters have vanished in the scuffle]_N⁶

8437 Relations that have multiple nuclei are called **coordinating**; relations with a single nu-
 8438 cleus are called **subordinating**. Subordinating relations are constrained to have only two
 8439 arguments, while coordinating relations (such as CONJUNCTION) may have more than
 8440 two.

8441 **RST Relations** Rhetorical structure theory features a large inventory of discourse rela-
 8442 tions, which are divided into two high-level groups: subject matter relations, and presen-
 8443 tational relations. Presentational relations are organized around the intended beliefs of
 8444 the reader. For example, in (16.8), the second discourse unit provides evidence intended
 8445 to increase the reader's belief in the proposition expressed by the first discourse unit, that
 8446 *LaShawn loves animals*. In contrast, subject-matter relations are meant to communicate ad-
 8447 ditional facts about the propositions contained in the discourse units that they relate:

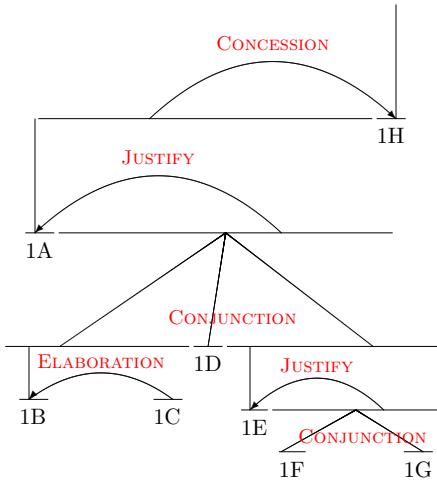
⁴Details of discourse segmentation can be found in the RST annotation manual (Carlson and Marcu, 2001).

⁵While RST analyses are typically trees, this should be taken as a strong theoretical commitment to the principle that all coherent discourses have a tree structure. Taboada and Mann (2006) write:

It is simply the case that trees are convenient, easy to represent, and easy to understand. There is, on the other hand, no theoretical reason to assume that trees are the only possible representation of discourse structure and of coherence relations.

The appropriateness of tree structures to discourse has been challenged, e.g., by Wolf and Gibson (2005), who propose a more general graph-structured representation.

⁶from the RST Treebank (Carlson et al., 2002)



[It could have been a great movie]^{1A} [It does have beautiful scenery,]^{1B} [some of the best since Lord of the Rings.]^{1C} [The acting is well done,]^{1D} [and I really liked the son of the leader of the Samurai.]^{1E} [He was a likable chap,]^{1F} [and I hated to see him die.]^{1G} [But, other than all that, this movie is nothing more than hidden rip-offs.]^{1H}

Figure 16.5: A rhetorical structure theory analysis of a short movie review, adapted from Voll and Taboada (2007). Positive and negative sentiment words are underlined, indicating RST's potential utility in document-level sentiment analysis.

8448 (16.10) [the debt plan was rushed to completion]_N
 8449 [in order to be announced at the meeting]_S⁷

8450 In this example, the satellite describes a world state that is realized by the action described
 8451 in the nucleus. This relationship is about the world, and not about the author's commu-
 8452 nicative intentions.

8453 **Example** Figure 16.5 depicts an RST analysis of a paragraph from a movie review. Asym-
 8454 metric (subordinating) relations are depicted with an arrow from the satellite to the nu-
 8455 cleus; symmetric (coordinating) relations are depicted with lines. The elementary dis-
 8456 course units 1F and 1G are combined into a larger discourse unit with the symmetric
 8457 CONJUNCTION relation. The resulting discourse unit is then the satellite in a JUSTIFY
 8458 relation with 1E.

⁷from the RST Treebank (Carlson et al., 2002)

8459 **16.3.2.1 Hierarchical discourse parsing**

8460 The goal of discourse parsing is to recover a hierarchical structural analysis from a doc-
 8461 ument text, such as the analysis in Figure 16.5. For now, let's assume a segmentation
 8462 of the document into elementary discourse units (EDUs); segmentation algorithms are
 8463 discussed below. After segmentation, discourse parsing can be viewed as a combination
 8464 of two components: the discourse relation classification techniques discussed in § 16.3.1.2,
 8465 and algorithms for phrase-structure parsing, such as chart parsing and shift-reduce, which
 8466 were discussed in chapter 10.

8467 Both chart parsing and shift-reduce require encoding composite discourse units, ei-
 8468 ther in a discrete feature vector or a dense neural representation.⁸ Some discourse parsers
 8469 rely on the **strong compositionality criterion** (Marcu, 1996), which states the assumption
 8470 that a composite discourse unit can be represented by its nucleus. This criterion is used in
 8471 feature-based discourse parsing to determine the feature vector for a composite discourse
 8472 unit (Hernault et al., 2010); it is used in neural approaches to setting the vector encod-
 8473 ing for a composite discourse unit equal to the encoding of its nucleus (Ji and Eisenstein,
 8474 2014). An alternative neural approach is to learn a composition function over the compo-
 8475 nents of a composite discourse unit (Li et al., 2014), using a recursive neural network (see
 8476 § 14.8.3).

8477 **Bottom-up discourse parsing** Assume a segmentation of the text into N elementary
 8478 discourse units with base representations $\{z^{(i)}\}_{i=1}^N$, and assume a composition function
 8479 $\text{COMPOSE}(z^{(i)}, z^{(j)}, \ell)$, which maps two encodings and a discourse relation ℓ into a new
 8480 encoding. The composition function can follow the strong compositionality criterion and
 8481 simply select the encoding of the nucleus, or it can do something more complex. We
 8482 also need a scoring function $\Psi(z^{(i,k)}, z^{(k,j)}, \ell)$, which computes a scalar score for the (bi-
 8483 narized) discourse relation ℓ with left child covering the span $i + 1 : k$, and the right
 8484 child covering the span $k + 1 : j$. Given these components, we can construct vector rep-
 8485 resentations for each span, and this is the basic idea underlying **compositional vector**
 8486 **grammars** (Socher et al., 2013).

8487 These same components can also be used in bottom-up parsing, in a manner that is
 8488 similar to the CKY algorithm for weighted context-free grammars (see § 10.1): compute
 8489 the score and best analysis for each possible span of increasing lengths, while storing
 8490 back-pointers that make it possible to recover the optimal parse of the entire input. How-
 8491 ever, there is an important distinction from CKY parsing: for each labeled span (i, j, ℓ) , we
 8492 must use the composition function to construct a representation $z^{(i,j,\ell)}$. This representa-
 8493 tion is then used to combine the discourse unit spanning $i + 1 : j$ in higher-level discourse
 8494 relations. The representation $z^{(i,j,\ell)}$ depends on the entire substructure of the unit span-

⁸To use these algorithms, is also necessary to binarize all discourse relations during parsing, and then to “unbinarize” them to reconstruct the desired structure (e.g., Hernault et al., 2010).

8495 ning $i + 1 : j$, and this violates the locality assumption that underlie CKY’s optimality
 8496 guarantee. Bottom-up parsing with recursively constructed span representations is gen-
 8497 erally not guaranteed to find the best-scoring discourse parse. This problem is explored
 8498 in an exercise at the end of the chapter.

8499 **Transition-based discourse parsing** One drawback of bottom-up parsing is its cubic
 8500 time complexity in the length of the input. For long documents, transition-based parsing
 8501 is an appealing alternative. The shift-reduce algorithm can be applied to discourse parsing
 8502 fairly directly (Sagae, 2009): the stack stores a set of discourse units and their repres-
 8503 entations, and each action is chosen by a function of these representations. This function
 8504 could be a linear product of weights and features, or it could be a neural network ap-
 8505 plied to encodings of the discourse units. The REDUCE action then performs composition
 8506 on the two discourse units at the top of the stack, yielding a larger composite discourse
 8507 unit, which goes on top of the stack. All of the techniques for integrating learning and
 8508 transition-based parsing, described in § 11.3, are applicable to discourse parsing.

8509 16.3.2.2 Segmenting discourse units

8510 In rhetorical structure theory, elementary discourse units do not cross the sentence bound-
 8511 ary, so discourse segmentation can be performed within sentences, assuming the sentence
 8512 segmentation is given. The segmentation of sentences into elementary discourse units is
 8513 typically performed using features of the syntactic analysis (Braud et al., 2017). One ap-
 8514 proach is to train a classifier to determine whether each syntactic constituent is an EDU,
 8515 using features such as the production, tree structure, and head words (Soricut and Marcu,
 8516 2003; Hernault et al., 2010). Another approach is to train a sequence labeling model, such
 8517 as a conditional random field (Sporleder and Lapata, 2005; Xuan Bach et al., 2012; Feng
 8518 et al., 2014). This is done using the BIO formalism for segmentation by sequence labeling,
 8519 described in § 8.3.

8520 16.3.3 Argumentation

8521 An alternative view of text-level relational structure focuses on **argumentation** (Stab and
 8522 Gurevych, 2014b). Each segment (typically a sentence or clause) may support or rebut
 8523 another segment, creating a graph structure over the text. In the following example (from
 8524 Peldszus and Stede, 2013), segment S_2 provides argumentative support for the proposi-
 8525 tion in the segment S_1 :

8526 (16.11) [We should tear the building down] $_{S1}$
 8527 [because it is full of asbestos] $_{S2}$.

8528 Assertions may also support or rebut proposed links between two other assertions, cre-
 8529 ating a **hypergraph**, which is a generalization of a graph to the case in which edges can

8530 join any number of vertices. This can be seen by introducing another sentence into the
8531 example:

8532 (16.12) [In principle it is possible to clean it up.]_{S3}
8533 [but according to the mayor that is too expensive.]_{S4}

8534 S3 acknowledges the validity of *S2*, but undercuts its support of *S1*. This can be repre-
8535 sented by introducing a hyperedge, $(S3, S2, S1)_{\text{undercut}}$, indicating that *S3* undercuts the
8536 proposed relationship between *S2* and *S1*. *S4* then undercuts the relevance of *S3*.

8537 **Argumentation mining** is the task of recovering such structures from raw texts. At
8538 present, annotations of argumentation structure are relatively small: Stab and Gurevych
8539 (2014a) have annotated a collection of 90 persuasive essays, and Peldszus and Stede (2015)
8540 have solicited and annotated a set of 112 paragraph-length “microtexts” in German.

8541 16.3.4 Applications of discourse relations

8542 The predominant application of discourse parsing is to select content within a document.
8543 In rhetorical structure theory, the nucleus is considered the more important element of
8544 the relation, and is more likely to be part of a summary of the document; it may also
8545 be more informative for document classification. The D-LTAG theory that underlies the
8546 Penn Discourse Treebank lacks this notion of nuclearity, but arguments may have varying
8547 importance, depending on the relation type. For example, the span of text constituting
8548 ARG1 of an expansion relation is more likely to appear in a summary, while the sentence
8549 constituting ARG2 of an implicit relation is less likely (Louis et al., 2010). Discourse relations
8550 may also signal segmentation points in the document structure. Explicit discourse
8551 markers have been shown to correlate with changes in subjectivity, and identifying such
8552 change points can improve document-level sentiment classification, by helping the clas-
8553 sifier to focus on the subjective parts of the text (Trivedi and Eisenstein, 2013; Yang and
8554 Cardie, 2014).

8555 16.3.4.1 Extractive Summarization

8556 Text **summarization** is the problem of converting a longer text into a shorter one, while
8557 still conveying the key facts, events, ideas, and sentiments from the original. In **extractive**
8558 **summarization**, the summary is a subset of the original text; in **abstractive summariza-**
8559 **tion**, the summary is produced *de novo*, by paraphrasing the original, or by first encoding
8560 it into a semantic representation (see § 19.2). The main strategy for extractive summa-
8561 rization is to maximize **coverage**, choosing a subset of the document that best covers the
8562 concepts mentioned in the document as a whole; typically, coverage is approximated by
8563 bag-of-words overlap (Nenkova and McKeown, 2012). Coverage-based objectives can be
8564 supplemented by hierarchical discourse relations, using the principle of nuclearity: in any
8565 subordinating discourse relation, the nucleus is more critical to the overall meaning of the

8566 text, and is therefore more important to include in an extractive summary (Marcu, 1997a).⁹
 8567 This insight can be generalized from individual relations using the concept of **discourse**
 8568 **depth** (Hirao et al., 2013): for each elementary discourse unit e , the discourse depth d_e is
 8569 the number of relations in which a discourse unit containing e is the satellite.

8570 Both discourse depth and nuclearity can be incorporated into extractive summarization
 8571 using constrained optimization. Let \mathbf{x}_n be a bag-of-words vector representation of
 8572 elementary discourse unit n , let $y_n \in \{0, 1\}$ indicate whether n is included in the summary,
 8573 and let d_n be the depth of unit n . Furthermore, let each discourse unit have a “head” h ,
 8574 which is defined recursively:

- 8575 • if a discourse unit is produced by a subordinating relation, then its head is the head
 8576 of the (unique) nucleus;
- 8577 • if a discourse unit is produced by a coordinating relation, then its head is the head
 8578 of the left-most nucleus;
- 8579 • for each elementary discourse unit, its parent $\pi(n) \in \{\emptyset, 1, 2, \dots, N\}$ is the head of
 8580 the smallest discourse unit containing n whose head is not n ;
- 8581 • if n is the head of the discourse unit spanning the whole document, then $\pi(n) = \emptyset$.

With these definitions in place, discourse-driven extractive summarization can be formalized as (Hirao et al., 2013),

$$\begin{aligned} & \max_{y=\{0,1\}^N} \sum_{n=1}^N y_n \frac{\Psi(\mathbf{x}_n, \{\mathbf{x}_{1:N}\})}{d_n} \\ & \text{s.t. } \sum_{n=1}^N y_n \left(\sum_{j=1}^V x_{n,j} \right) \leq L \\ & \quad y_{\pi(n)} \geq y_n, \quad \forall n \end{aligned} \tag{16.11}$$

8582 where $\Psi(\mathbf{x}_n, \{\mathbf{x}_{1:N}\})$ measures the coverage of elementary discourse unit n with respect
 8583 to the rest of the document, and $\sum_{j=1}^V x_{n,m}$ is the number of tokens in \mathbf{x}_n . The first con-
 8584 straint ensures that the number of tokens in the summary has an upper bound L . The
 8585 second constraint ensures that no elementary discourse unit is included unless its parent
 8586 is also included. In this way, the discourse structure is used twice: to downweight the
 8587 contributions of elementary discourse units that are not central to the discourse, and to
 8588 ensure that the resulting structure is a subtree of the original discourse parse. The opti-

⁹Conversely, the arguments of a multi-nuclear relation should either both be included in the summary, or both excluded (Durrett et al., 2016).

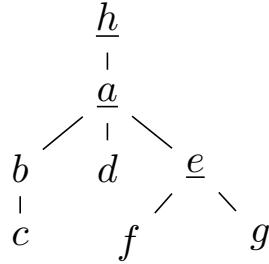


Figure 16.6: A **discourse depth tree** (Hirao et al., 2013) for the discourse parse from Figure 16.5, in which each elementary discourse unit is connected to its parent. The discourse units in one valid summary are underlined.

8589 mization problem in 16.11 can be solved with **integer linear programming**, described in
 8590 § 13.2.2.¹⁰

8591 Figure 16.6 shows a **discourse depth tree** for the RST analysis from Figure 16.5, in
 8592 which each elementary discourse is connected to (and below) its parent. The figure also
 8593 shows a valid summary, corresponding to:

8594 (16.13) It could have been a great movie, and I really liked the son of the leader of the
 8595 Samurai. But, other than all that, this movie is nothing more than hidden rip-offs.

8596 16.3.4.2 Document classification

8597 Hierarchical discourse structures lend themselves naturally to text classification: in a sub-
 8598 ordinating discourse relation, the nucleus should play a stronger role in the classification
 8599 decision than the satellite. Various implementations of this idea have been proposed.

- 8600 • Focusing on within-sentence discourse relations and lexicon-based classification (see
 8601 § 4.1.2), Voll and Taboada (2007) simply ignore the text in the satellites of each dis-
 8602 course relation.
- 8603 • At the document level, elements of each discourse relation argument can be reweighted,
 8604 favoring words in the nucleus, and disfavoring words in the satellite (Heerschop
 8605 et al., 2011; Bhatia et al., 2015). This approach can be applied recursively, computing
 8606 weights across the entire document. The weights can be relation-specific, so that the
 8607 features from the satellites of contrastive relations are discounted or even reversed.
- 8608 • Alternatively, the hierarchical discourse structure can define the structure of a **re-
 8609 cursive neural network** (see § 10.6.1). In this network, the representation of each

¹⁰Formally, 16.11 is a special case of the **knapsack problem**, in which the goal is to find a subset of items with maximum value, constrained by some maximum weight (Cormen et al., 2009).

8610 discourse unit is computed from its arguments and from a parameter corresponding
 8611 to the discourse relation (Ji and Smith, 2017).

8612 Shallow, non-hierarchical discourse relations have also been applied to document clas-
 8613 sification. One approach is to impose a set of constraints on the analyses of individual
 8614 discourse units, so that adjacent units have the same polarity when they are connected
 8615 by a discourse relation indicating agreement, and opposite polarity when connected by a
 8616 contrastive discourse relation, indicating disagreement (Somasundaran et al., 2009; Zirn
 8617 et al., 2011). Yang and Cardie (2014) apply explicitly-marked relations from the Penn
 8618 Discourse Treebank to the problem of sentence-level sentiment polarity classification (see
 8619 § 4.1). They impose the following soft constraints:

- 8620 • When a CONTRAST relation appears between two sentences, those sentences should
 8621 have opposite sentiment polarity.
- 8622 • When an EXPANSION or CONTINGENCY relation appears between two sentences,
 8623 they should have the same polarity.
- 8624 • When a CONTRAST relation appears *within* a sentence, it should have neutral polar-
 8625 ity, since it is likely to express both sentiments.

8626 These discourse-driven constraints are shown to improve performance on two datasets of
 8627 product reviews.

8628 16.3.4.3 Coherence

8629 Just as **grammaticality** is the property shared by well-structured sentences, **coherence** is
 8630 the property shared by well-structured discourses. One application of discourse process-
 8631 ing is to measure (and maximize) the coherence of computer-generated texts like transla-
 8632 tions and summaries (Kibble and Power, 2004). Coherence assessment is also used to eval-
 8633 uate human-generated texts, such as student essays (e.g., Miltsakaki and Kukich, 2004;
 8634 Burstein et al., 2013).

8635 Coherence subsumes a range of phenomena, many of which have been highlighted
 8636 earlier in this chapter: e.g., that adjacent sentences should be lexically cohesive (Foltz
 8637 et al., 1998; Ji et al., 2015; Li and Jurafsky, 2017), and that entity references should follow
 8638 the principles of centering theory (Barzilay and Lapata, 2008; Nguyen and Joty, 2017).
 8639 Discourse relations also bear on the coherence of a text in a variety of ways:

- 8640 • Hierarchical discourse relations tend to have a “canonical ordering” of the nucleus
 8641 and satellite (Mann and Thompson, 1988): for example, in the ELABORATION rela-
 8642 tion from rhetorical structure theory, the nucleus always comes first, while in the
 8643 JUSTIFICATION relation, the satellite tends to be first (Marcu, 1997b).

- Discourse relations should be signaled by connectives that are appropriate to the semantic or functional relationship between the arguments: for example, a coherent text would be more likely to use *however* to signal a COMPARISON relation than a *temporal* relation (Kibble and Power, 2004).
- Discourse relations tend to appear in predictable sequences: for example, COMPARISON relations tend to immediately precede CONTINGENCY relations (Pitler et al., 2008). This observation can be formalized by generalizing the entity grid model (§ 16.2.2), so that each cell (i, j) provides information about the role of the discourse argument containing a mention of entity j in sentence i (Lin et al., 2011). For example, if the first sentence is ARG1 of a comparison relation, then any entity mentions in the sentence would be labeled COMP.ARG1. This approach can also be applied to RST discourse relations (Feng et al., 2014).

Datasets One difficulty with evaluating metrics of discourse coherence is that human-generated texts usually meet some minimal threshold of coherence. For this reason, much of the research on measuring coherence has focused on synthetic data. A typical setting is to permute the sentences of a human-written text, and then determine whether the original sentence ordering scores higher according to the proposed coherence measure (Barzilay and Lapata, 2008). There are also small datasets of human evaluations of the coherence of machine summaries: for example, human judgments of the summaries from the participating systems in the 2003 Document Understanding Conference are available online.¹¹ Researchers from the Educational Testing Service (an organization which administers several national exams in the United States) have studied the relationship between discourse coherence and student essay quality (Burstein et al., 2003, 2010). A public dataset of essays from second-language learners, with quality annotations, has been made available by researchers at Cambridge University (Yannakoudakis et al., 2011). At the other extreme, Louis and Nenkova (2013) analyze the structure of professionally written scientific essays, finding that discourse relation transitions help to distinguish prize-winning essays from other articles in the same genre.

Additional resources

For a manuscript-length discussion of discourse processing, see Stede (2011). Article-length surveys are offered by Webber et al. (2012) and Webber and Joshi (2012).

¹¹<http://homepages.inf.ed.ac.uk/mlap/coherence/>

8675 **Exercises**

- 8676 1. • Implement the smoothed cosine similarity metric from Equation 16.2, using the
 smoothing kernel $k = [.5, .3, .15, .05]$.
 • Download the text of a news article with at least ten paragraphs.
 • Compute and plot the smoothed similarity \bar{s} over the length of the article.
 • Identify *local minima* in \bar{s} as follows: first find all sentences m such that $\bar{s}_m < \bar{s}_{m\pm1}$. Then search among these points to find the five sentences with the lowest
 \bar{s}_m .
 • How often do the five local minima correspond to paragraph boundaries?
 – The fraction of local minima that are paragraph boundaries is the **precision-at- k** , where in this case, $k = 5$.
 – The fraction of paragraph boundaries which are local minima is the **recall-at- k** .
 – Compute precision-at- k and recall-at- k for $k = 3$ and $k = 10$.
- 8684 2. This exercise is to be done in pairs. Each participant selects an article from to-
 day's news, and replaces all mentions of individual people with special tokens like
 PERSON1, PERSON2, and so on. The other participant should then use the rules
 of centering theory to guess each type of referring expression: full name (*Captain*
 Ahab), partial name (e.g., *Ahab*), nominal (e.g., *the ship's captain*), or pronoun. Check
 whether the predictions match the original article, and whether the original article
 conforms to the rules of centering theory.
- 8696 3. In § 16.3.2.1, it is noted that bottom-up parsing with compositional representations
 of each span is not guaranteed to be optimal. In this exercise, you will construct
 a minimal example proving this point. Consider a discourse with four units, with
 base representations $\{z^{(i)}\}_{i=1}^4$. Construct a scenario in which the parse selected by
 bottom-up parsing is not optimal, and give the precise mathematical conditions that
 must hold for this suboptimal parse to be selected. You may ignore the relation
 labels ℓ for the purpose of this example.

8703

Part IV

8704

Applications

405

8705 Chapter 17

8706 Information extraction

8707 Computers offer powerful capabilities for searching and reasoning about structured records
8708 and relational data. Some even argue that the most important limitation of artificial intel-
8709 ligence is not inference or learning, but simply having too little knowledge (Lenat et al.,
8710 1990). Natural language processing provides an appealing solution: automatically con-
8711 struct a structured **knowledge base** by reading natural language text.

8712 For example, many Wikipedia pages have an “infobox” that provides structured in-
8713 formation about an entity or event. An example is shown in Figure 17.1a: each row rep-
8714 resents one or more properties of the entity IN THE AEROPLANE OVER THE SEA, a record
8715 album. The set of properties is determined by a predefined **schema**, which applies to all
8716 record albums in Wikipedia. As shown in Figure 17.1b, the values for many of these fields
8717 are indicated directly in the first few sentences of text on the same Wikipedia page.

8718 The task of automatically constructing (or “populating”) an infobox from text is an
8719 example of **information extraction**. Much of information extraction can be described in
8720 terms of **entities**, **relations**, and **events**.

- 8721 • **Entities** are uniquely specified objects in the world, such as people (JEFF MANGUM),
8722 places (ATHENS, GEORGIA), organizations (MERGE RECORDS), and times (FEBRUARY
8723 10, 1998). Chapter 8 described the task of **named entity recognition**, which labels
8724 tokens as parts of entity spans. Now we will see how to go further, **linking** each
8725 entity **mention** to an element in a **knowledge base**.
- 8726 • **Relations** include a **predicate** and two **arguments**: for example, CAPITAL(GEORGIA, ATLANTA).
- **Events** involve multiple typed arguments. For example, the production and release

Studio album by Neutral Milk Hotel	
Released	February 10, 1998
Recorded	July–September 1997
Studio	Pet Sounds Studio, Denver, Colorado
Genre	Indie rock • psychedelic folk • lo-fi
Length	39:55
Label	Merge • Domino
Producer	Robert Schneider

(a) A Wikipedia infobox

- (17.1) In the Aeroplane Over the Sea is the second and final studio album by the American indie rock band Neutral Milk Hotel.
- (17.2) It was released in the United States on February 10, 1998 on Merge Records and May 1998 on Blue Rose Records in the United Kingdom.
- (17.3) Jeff Mangum moved from Athens, Georgia to Denver, Colorado to prepare the bulk of the album's material with producer Robert Schneider, this time at Schneider's newly created Pet Sounds Studio at the home of Jim McIntyre.

- (b) The first few sentences of text. Strings that match fields or field names in the infobox are underlined; strings that mention other entities are wavy underlined.

Figure 17.1: From the Wikipedia page for the album “In the Aeroplane Over the Sea”, retrieved October 26, 2017.

of the album described in Figure 17.1 is described by the event,

```
<TITLE : IN THE AEROPLANE OVER THE SEA,
ARTIST : NEUTRAL MILK HOTEL,
RELEASE-DATE : 1998-FEB-10,...>
```

8727 The set of arguments for an event type is defined by a **schema**. Events often refer to
 8728 time-delimited occurrences: weddings, protests, purchases, terrorist attacks.

8729 Information extraction is similar to semantic role labeling (chapter 13): we may think
 8730 of predicates as corresponding to events, and the arguments as defining slots in the event
 8731 representation. However, the goals of information extraction are different. Rather than
 8732 accurately parsing every sentence, information extraction systems often focus on recog-
 8733 nizing a few key relation or event types, or on the task of identifying all properties of a
 8734 given entity. Information extraction is often evaluated by the correctness of the resulting
 8735 knowledge base, and not by how many sentences were accurately parsed. The goal is
 8736 sometimes described as **macro-reading**, as opposed to **micro-reading**, in which each sen-
 8737 tence must be analyzed correctly. Macro-reading systems are not penalized for ignoring
 8738 difficult sentences, as long as they can recover the same information from other, easier-
 8739 to-read sources. However, macro-reading systems must resolve apparent inconsistencies

8740 (was the album released on MERGE RECORDS or BLUE ROSE RECORDS?), requiring reasoning across the entire dataset.

8742 In addition to the basic tasks of recognizing entities, relations, and events, information
8743 extraction systems must handle negation, and must be able to distinguish statements of
8744 fact from hopes, fears, hunches, and hypotheticals. Finally, information extraction is often paired with the problem of **question answering**, which requires accurately parsing a
8746 query, and then selecting or generating a textual answer. Question answering systems can
8747 be built on knowledge bases that are extracted from large text corpora, or may attempt to
8748 identify answers directly from the source texts.

8749 17.1 Entities

8750 The starting point for information extraction is to identify mentions of entities in text.
8751 Consider the following example:

8752 (17.4) *The United States Army captured a hill overlooking Atlanta on May 14, 1864.*

8753 For this sentence, there are two goals:

- 8754 1. *Identify* the spans *United States Army*, *Atlanta*, and *May 14, 1864* as entity mentions.
8755 (The hill is not uniquely identified, so it is not a *named* entity.) We may also want to
8756 recognize the **named entity types**: organization, location, and date. This is **named**
8757 **entity recognition**, and is described in chapter 8.
- 8758 2. *Link* these spans to entities in a knowledge base: U.S. ARMY, ATLANTA, and 1864-
8759 MAY-14. This task is known as **entity linking**.

8760 The strings to be linked to entities are **mentions** — similar to the use of this term in
8761 coreference resolution. In some formulations of the entity linking task, only named entities
8762 are candidates for linking. This is sometimes called **named entity linking** (Ling et al.,
8763 2015). In other formulations, such as **Wikification** (Milne and Witten, 2008), any string
8764 can be a mention. The set of target entities often corresponds to Wikipedia pages, and
8765 Wikipedia is the basis for more comprehensive knowledge bases such as YAGO (Suchanek
8766 et al., 2007), DBPedia (Auer et al., 2007), and Freebase (Bollacker et al., 2008). Entity link-
8767 ing may also be performed in more “closed” settings, where a much smaller list of targets
8768 is provided in advance. The system must also determine if a mention does not refer to
8769 any entity in the knowledge base, sometimes called a **NIL entity** (McNamee and Dang,
8770 2009).

8771 Returning to (17.4), the three entity mentions may seem unambiguous. But the Wikipedia
8772 disambiguation page for the string *Atlanta* says otherwise:¹ there are more than twenty

¹[https://en.wikipedia.org/wiki/Atlanta_\(disambiguation\)](https://en.wikipedia.org/wiki/Atlanta_(disambiguation)), retrieved November 1, 2017.

8773 different towns and cities, five United States Navy vessels, a magazine, a television show,
 8774 a band, and a singer — each prominent enough to have its own Wikipedia page. We now
 8775 consider how to choose among these dozens of possibilities. In this chapter we will focus
 8776 on supervised approaches. Unsupervised entity linking is closely related to the problem
 8777 of **cross-document coreference resolution**, where the task is to identify pairs of mentions
 8778 that corefer, across document boundaries (Bagga and Baldwin, 1998b; Singh et al., 2011).

8779 17.1.1 Entity linking by learning to rank

8780 Entity linking is often formulated as a **ranking** problem,

$$\hat{y} = \operatorname{argmax}_{y \in \mathcal{Y}(x)} \Psi(y, x, c), \quad [17.1]$$

8781 where y is a target entity, x is a description of the mention, $\mathcal{Y}(x)$ is a set of candidate
 8782 entities, and c is a description of the context — such as the other text in the document,
 8783 or its metadata. The function Ψ is a scoring function, which could be a linear model,
 8784 $\Psi(y, x, c) = \theta \cdot f(y, x, c)$, or a more complex function such as a neural network. In either
 8785 case, the scoring function can be learned by minimizing a margin-based **ranking loss**,

$$\ell(\hat{y}, y^{(i)}, x^{(i)}, c^{(i)}) = (\Psi(\hat{y}, x^{(i)}, c^{(i)}) - \Psi(y^{(i)}, x^{(i)}, c^{(i)}) + 1)_+, \quad [17.2]$$

8786 where $y^{(i)}$ is the ground truth and $\hat{y} \neq y^{(i)}$ is the predicted target for mention $x^{(i)}$ in
 8787 context $c^{(i)}$ (Joachims, 2002; Dredze et al., 2010).

8788 **Candidate identification** For computational tractability, it is helpful to restrict the set of
 8789 candidates, $\mathcal{Y}(x)$. One approach is to use a **name dictionary**, which maps from strings
 8790 to the entities that they might mention. This mapping is many-to-many: a string such as
 8791 *Atlanta* can refer to multiple entities, and conversely, an entity such as ATLANTA can be
 8792 referenced by multiple strings. A name dictionary can be extracted from Wikipedia, with
 8793 links between each Wikipedia entity page and the anchor text of all hyperlinks that point
 8794 to the page (Bunescu and Pasca, 2006; Ratinov et al., 2011). To improve recall, the name
 8795 dictionary can be augmented by partial and approximate matching (Dredze et al., 2010),
 8796 but as the set of candidates grows, the risk of false positives increases. For example, the
 8797 string *Atlanta* is a partial match to *the Atlanta Fed* (a name for the FEDERAL RESERVE BANK
 8798 OF ATLANTA), and a noisy match (edit distance of one) from *Atalanta* (a heroine in Greek
 8799 mythology and an Italian soccer team).

8800 **Features** Feature-based approaches to entity ranking rely on three main types of local
 8801 information (Dredze et al., 2010):

- The similarity of the mention string to the canonical entity name, as quantified by string similarity. This feature would elevate the city ATLANTA over the basketball team ATLANTA HAWKS for the string *Atlanta*.
- The popularity of the entity, which can be measured by Wikipedia page views or PageRank in the Wikipedia link graph. This feature would elevate ATLANTA, GEORGIA over the unincorporated community of ATLANTA, OHIO.
- The entity type, as output by the named entity recognition system. This feature would elevate the city of ATLANTA over the magazine ATLANTA in contexts where the mention is tagged as a location.

In addition to these local features, the document context can also help. If *Jamaica* is mentioned in a document about the Caribbean, it is likely to refer to the island nation; in the context of New York, it is likely to refer to the neighborhood in Queens; in the context of a menu, it might refer to a hibiscus tea beverage. Such hints can be formalized by computing the similarity between the Wikipedia page describing each candidate entity and the mention context $c^{(i)}$, which may include the bag-of-words representing the document (Dredze et al., 2010; Hoffart et al., 2011) or a smaller window of text around the mention (Ratinov et al., 2011). For example, we can compute the cosine similarity between bag-of-words vectors for the context and entity description, typically weighted using **inverse document frequency** to emphasize rare words.²

Neural entity linking An alternative approach is to compute the score for each entity candidate using distributed vector representations of the entities, mentions, and context. For example, for the task of entity linking in Twitter, Yang et al. (2016) employ the bilinear scoring function,

$$\Psi(y, x, c) = v_y^\top \Theta^{(y,x)} x + v_y^\top \Theta^{(y,c)} c, \quad [17.3]$$

with $v_y \in \mathbb{R}^{K_y}$ as the vector embedding of entity y , $x \in \mathbb{R}^{K_x}$ as the embedding of the mention, $c \in \mathbb{R}^{K_c}$ as the embedding of the context, and the matrices $\Theta^{(y,x)}$ and $\Theta^{(y,c)}$ as parameters that score the compatibility of each entity with respect to the mention and context. Each of the vector embeddings can be learned from an end-to-end objective, or pre-trained on unlabeled data.

- Pretrained **entity embeddings** can be obtained from an existing knowledge base (Bordes et al., 2011, 2013), or by running a word embedding algorithm such as WORD2VEC

²The **document frequency** of word j is $DF(j) = \frac{1}{N} \sum_{i=1}^N \delta(x_j^{(i)} > 0)$, equal to the number of documents in which the word appears. The contribution of each word to the cosine similarity of two bag-of-words vectors can be weighted by the **inverse document frequency** $\frac{1}{DF(j)}$ or $\log \frac{1}{DF(j)}$, to emphasize rare words (Spärck Jones, 1972).

- 8832 on the text of Wikipedia, with hyperlinks substituted for the anchor text.³
- 8833 • The embedding of the mention x can be computed by averaging the embeddings
 8834 of the words in the mention (Yang et al., 2016), or by the compositional techniques
 8835 described in § 14.8.
- 8836 • The embedding of the context c can also be computed from the embeddings of the
 8837 words in the context. A **denoising autoencoder** learns a function from raw text to
 8838 dense K -dimensional vector encodings by minimizing a reconstruction loss (Vin-
 8839 cent et al., 2010),

$$\min_{\theta_g, \theta_h} \sum_{i=1}^N \|\mathbf{x}^{(i)} - g(h(\tilde{\mathbf{x}}^{(i)}; \theta_h); \theta_g)\|^2, \quad [17.4]$$

8840 where $\tilde{\mathbf{x}}^{(i)}$ is a noisy version of the bag-of-words counts $\mathbf{x}^{(i)}$, which is produced by
 8841 randomly setting some counts to zero; $h : \mathbb{R}^V \mapsto \mathbb{R}^K$ is an encoder with parameters
 8842 θ_h ; and $g : \mathbb{R}^K \mapsto \mathbb{R}^V$, with parameters θ_g . The encoder and decoder functions
 8843 are typically implemented as feedforward neural networks. To apply this model to
 8844 entity linking, each entity and context are initially represented by the encoding of
 8845 their bag-of-words vectors, $h(e)$ and $g(c)$, and these encodings are then fine-tuned
 8846 from labeled data (He et al., 2013). The context vector c can also be obtained by
 8847 convolution on the embeddings of words in the document (Sun et al., 2015), or by
 8848 examining metadata such as the author’s social network (Yang et al., 2016).

8849 The remaining parameters $\Theta^{(y,x)}$ and $\Theta^{(y,c)}$ can be trained by backpropagation from the
 8850 margin loss in Equation 17.2.

8851 17.1.2 Collective entity linking

8852 Entity linking can be more accurate when it is performed jointly across a document. To
 8853 see why, consider the following lists:

- 8854 (17.5) California, Oregon, Washington
 8855 (17.6) Baltimore, Washington, Philadelphia
 8856 (17.7) Washington, Adams, Jefferson

8857 In each case, the term *Washington* refers to a different entity, and this reference is strongly
 8858 suggested by the other entries on the list. In the last list, all three names are highly am-
 8859 biguous — there are dozens of other *Adams* and *Jefferson* entities in Wikipedia. But a

³Pre-trained entity embeddings can be downloaded from <https://code.google.com/archive/p/word2vec/>.

8856 preference for coherence motivates **collectively** linking these references to the first three
 8857 U.S. presidents.

8858 A general approach to collective entity linking is to introduce a compatibility score
 8859 $\psi_c(\mathbf{y})$. Collective entity linking is then performed by optimizing the global objective,

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathbb{Y}(\mathbf{x})}{\operatorname{argmax}} \Psi_c(\mathbf{y}) + \sum_{i=1}^N \Psi_\ell(y^{(i)}, \mathbf{x}^{(i)}, \mathbf{c}^{(i)}), \quad [17.5]$$

8860 where $\mathbb{Y}(\mathbf{x})$ is the set of all possible collective entity assignments for the mentions in \mathbf{x} ,
 8861 and ψ_ℓ is the local scoring function for each entity i . The compatibility function is typically
 8862 decomposed into a sum of pairwise scores, $\Psi_c(\mathbf{y}) = \sum_{i=1}^N \sum_{j \neq i}^N \Psi_c(y^{(i)}, y^{(j)})$. These scores
 8863 can be computed in a number of different ways:

- 8864 • Wikipedia defines high-level categories for entities (e.g., *living people*, *Presidents of*
 8865 *the United States*, *States of the United States*), and Ψ_c can reward entity pairs for the
 8866 number of categories that they have in common (Cucerzan, 2007).
- 8867 • Compatibility can be measured by the number of incoming hyperlinks shared by
 8868 the Wikipedia pages for the two entities (Milne and Witten, 2008).
- 8869 • In a neural architecture, the compatibility of two entities can be set equal to the inner
 8870 product of their embeddings, $\Psi_c(y^{(i)}, y^{(j)}) = \mathbf{v}_{y^{(i)}} \cdot \mathbf{v}_{y^{(j)}}$.
- 8871 • A non-pairwise compatibility score can be defined using a type of latent variable
 8872 model known as a **probabilistic topic model** (Blei et al., 2003; Blei, 2012). In this
 8873 framework, each latent topic is a probability distribution over entities, and each
 8874 document has a probability distribution over topics. Each entity helps to determine
 8875 the document's distribution over topics, and in turn these topics help to resolve am-
 8876 biguous entity mentions (Newman et al., 2006). Inference can be performed using
 8877 the sampling techniques described in chapter 5.

8878 Unfortunately, collective entity linking is **NP-hard** even for pairwise compatibility func-
 8879 tions, so exact optimization is almost certainly intractable. Various approximate inference
 8880 techniques have been proposed, including **integer linear programming** (Cheng and Roth,
 8881 2013), **Gibbs sampling** (Han and Sun, 2012), and graph-based algorithms (Hoffart et al.,
 8882 2011; Han et al., 2011).

8883 17.1.3 *Pairwise ranking loss functions

8884 The loss function defined in Equation 17.2 considers only the highest-scoring prediction
 8885 \hat{y} , but in fact, the true entity $y^{(i)}$ should outscore *all* other entities. A loss function based on
 8886 this idea would give a gradient against the features or representations of several entities,

Algorithm 19 WARP approximate ranking loss

```

1: procedure WARP( $y^{(i)}$ ,  $\mathbf{x}^{(i)}$ )
2:    $N \leftarrow 0$ 
3:   repeat
4:     Randomly sample  $y \sim \mathcal{Y}(\mathbf{x}^{(i)})$ 
5:      $N \leftarrow N + 1$ 
6:     if  $\psi(y, \mathbf{x}^{(i)}) + 1 > \psi(y^{(i)}, \mathbf{x}^{(i)})$  then            $\triangleright$  check for margin violation
7:        $r \leftarrow \lfloor |\mathcal{Y}(\mathbf{x}^{(i)})|/N \rfloor$                           $\triangleright$  compute approximate rank
8:       return  $L_{\text{rank}}(r) \times (\psi(y, \mathbf{x}^{(i)}) + 1 - \psi(y^{(i)}, \mathbf{x}^{(i)}))$ 
9:     until  $N \geq |\mathcal{Y}(\mathbf{x}^{(i)})| - 1$                             $\triangleright$  no violation found
10:    return 0                                          $\triangleright$  return zero loss

```

8887 not just the top-scoring prediction. Usunier et al. (2009) define a general ranking error
 8888 function,

$$L_{\text{rank}}(k) = \sum_{j=1}^k \alpha_j, \quad \text{with } \alpha_1 \geq \alpha_2 \geq \dots \geq 0, \quad [17.6]$$

8889 where k is equal to the number of labels ranked higher than the correct label $y^{(i)}$. This
 8890 function defines a class of ranking errors: if $\alpha_j = 1$ for all j , then the ranking error is
 8891 equal to the rank of the correct entity; if $\alpha_1 = 1$ and $\alpha_{j>1} = 0$, then the ranking error is
 8892 one whenever the correct entity is not ranked first; if α_j decreases smoothly with j , as in
 8893 $\alpha_j = \frac{1}{j}$, then the error is between these two extremes.

This ranking error can be integrated into a margin objective. Remember that large margin classification requires not only the correct label, but also that the correct label outscores other labels by a substantial margin. A similar principle applies to ranking: we want a high rank for the correct entity, and we want it to be separated from other entities by a substantial margin. We therefore define the margin-augmented rank,

$$r(y^{(i)}, \mathbf{x}^{(i)}) \triangleq \sum_{y \in \mathcal{Y}(\mathbf{x}^{(i)}) \setminus y^{(i)}} \delta \left(1 + \psi(y, \mathbf{x}^{(i)}) \geq \psi(y^{(i)}, \mathbf{x}^{(i)}) \right), \quad [17.7]$$

8894 where $\delta(\cdot)$ is a delta function, and $\mathcal{Y}(\mathbf{x}^{(i)}) \setminus y^{(i)}$ is the set of all entity candidates minus
 8895 the true entity $y^{(i)}$. The margin-augmented rank is the rank of the true entity, after aug-
 8896 menting every other candidate with a margin of one, under the current scoring function
 8897 ψ . (The context c is omitted for clarity, and can be considered part of x .)

For each instance, a hinge loss is computed from the ranking error associated with this

margin-augmented rank, and the violation of the margin constraint,

$$\ell(y^{(i)}, \mathbf{x}^{(i)}) = \frac{L_{\text{rank}}(r(y^{(i)}, \mathbf{x}^{(i)}))}{r(y^{(i)}, \mathbf{x}^{(i)})} \sum_{y \in \mathcal{Y}(\mathbf{x}) \setminus y^{(i)}} \left(\psi(y, \mathbf{x}^{(i)}) - \psi(y^{(i)}, \mathbf{x}^{(i)}) + 1 \right)_+, \quad [17.8]$$

8898 The sum in Equation 17.8 includes non-zero values for every label that is ranked at least as
 8899 high as the true entity, after applying the margin augmentation. Dividing by the margin-
 8900 augmented rank of the true entity thus gives the average violation.

8901 The objective in Equation 17.8 is expensive to optimize when the label space is large,
 8902 as is usually the case for entity linking against large knowledge bases. This motivates a
 8903 randomized approximation called **WARP** (Weston et al., 2011), shown in Algorithm 19. In
 8904 this procedure, we sample random entities until one violates the pairwise margin con-
 8905 straint, $\psi(y, \mathbf{x}^{(i)}) + 1 \geq \psi(y^{(i)}, \mathbf{x}^{(i)})$. The number of samples N required to find such
 8906 a violation yields an approximation of the margin-augmented rank of the true entity,
 8907 $r(y^{(i)}, \mathbf{x}^{(i)}) \approx \left\lfloor \frac{|\mathcal{Y}(\mathbf{x})|}{N} \right\rfloor$. If a violation is found immediately, $N = 1$, the correct entity
 8908 probably ranks below many others, $r \approx |\mathcal{Y}(\mathbf{x})|$. If many samples are required before a
 8909 violation is found, $N \rightarrow |\mathcal{Y}(\mathbf{x})|$, then the correct entity is probably highly ranked, $r \rightarrow 1$.
 8910 A computational advantage of WARP is that it is not necessary to find the highest-scoring
 8911 label, which can impose a non-trivial computational cost when $\mathcal{Y}(\mathbf{x}^{(i)})$ is large. The objec-
 8912 tive is conceptually similar to the **negative sampling** objective in WORD2VEC (chapter 14),
 8913 which compares the observed word against randomly sampled alternatives.

8914 17.2 Relations

8915 After identifying the entities that are mentioned in a text, the next step is to determine
 8916 how they are related. Consider the following example:

8917 (17.8) George Bush traveled to France on Thursday for a summit.

8918 This sentence introduces a relation between the entities referenced by *George Bush* and
 8919 *France*. In the Automatic Content Extraction (ACE) ontology (Linguistic Data Consortium,
 8920 2005), the type of this relation is PHYSICAL, and the subtype is LOCATED. This relation
 8921 would be written,

$$\text{PHYSICAL.LOCATED(GEORGE BUSH, FRANCE)}. \quad [17.9]$$

8922 Relations take exactly two arguments, and the order of the arguments matters.

8923 In the ACE datasets, relations are annotated between entity mentions, as in the exam-
 8924 ple above. Relations can also hold between nominals, as in the following example from
 8925 the SemEval-2010 shared task (Hendrickx et al., 2009):

CAUSE-EFFECT	<i>those cancers were caused by radiation exposures</i>
INSTRUMENT-AGENCY	<i>phone operator</i>
PRODUCT-PRODUCER	<i>a factory manufactures suits</i>
CONTENT-CONTAINER	<i>a bottle of honey was weighed</i>
ENTITY-ORIGIN	<i>letters from foreign countries</i>
ENTITY-DESTINATION	<i>the boy went to bed</i>
COMPONENT-WHOLE	<i>my apartment has a large kitchen</i>
MEMBER-COLLECTION	<i>there are many trees in the forest</i>
COMMUNICATION-TOPIC	<i>the lecture was about semantics</i>

Table 17.1: Relations and example sentences from the SemEval-2010 dataset (Hendrickx et al., 2009)

8926 (17.9) The cup contained tea from dried ginseng.

8927 This sentence describes a relation of type ENTITY-ORIGIN between *tea* and *ginseng*. Nominal
 8928 relation extraction is closely related to **semantic role labeling** (chapter 13). The main
 8929 difference is that relation extraction is restricted to a relatively small number of relation
 8930 types; for example, Table 17.1 shows the ten relation types from SemEval-2010.

8931 17.2.1 Pattern-based relation extraction

8932 Early work on relation extraction focused on hand-crafted patterns (Hearst, 1992). For
 8933 example, the appositive *Starbuck, a native of Nantucket* signals the relation ENTITY-ORIGIN
 8934 between *Starbuck* and *Nantucket*. This pattern can be written as,

$$\text{PERSON , } a \text{ native of LOCATION} \Rightarrow \text{ENTITY-ORIGIN(PERSON, LOCATION)}. \quad [17.10]$$

8935 This pattern will be “triggered” whenever the literal string *, a native of* occurs between an
 8936 entity of type PERSON and an entity of type LOCATION. Such patterns can be generalized
 8937 beyond literal matches using techniques such as lemmatization, which would enable the
 8938 words (*buy, buys, buying*) to trigger the same patterns (see § 4.3.1.2). A more aggressive
 8939 strategy would be to group all words in a WordNet synset (§ 4.2), so that, e.g., *buy* and
 8940 *purchase* trigger the same patterns.

8941 Relation extraction patterns can be implemented in finite-state automata (§ 9.1). If the
 8942 named entity recognizer is also a finite-state machine, then the systems can be combined
 8943 by finite-state transduction (Hobbs et al., 1997). This makes it possible to propagate uncer-
 8944 tainty through the finite-state cascade, and disambiguate from higher-level context. For
 8945 example, suppose the entity recognizer cannot decide whether *Starbuck* refers to either a
 8946 PERSON or a LOCATION; in the composed transducer, the relation extractor would be free
 8947 to select the PERSON annotation when it appears in the context of an appropriate pattern.

8948 **17.2.2 Relation extraction as a classification task**

8949 Relation extraction can be formulated as a classification problem,

$$\hat{r}_{(i,j),(m,n)} = \operatorname{argmax}_{r \in \mathcal{R}} \Psi(r, (i, j), (m, n), \mathbf{w}), \quad [17.11]$$

8950 where $r \in \mathcal{R}$ is a relation type (possibly NIL), $\mathbf{w}_{i+1:j}$ is the span of the first argument, and
 8951 $\mathbf{w}_{m+1:n}$ is the span of the second argument. The argument $\mathbf{w}_{m+1:n}$ may appear before
 8952 or after $\mathbf{w}_{i+1:j}$ in the text, or they may overlap; we stipulate only that $\mathbf{w}_{i+1:j}$ is the first
 8953 argument of the relation. We now consider three alternatives for computing the scoring
 8954 function.

8955 **17.2.2.1 Feature-based classification**

8956 In a feature-based classifier, the scoring function is defined as,

$$\Psi(r, (i, j), (m, n), \mathbf{w}) = \boldsymbol{\theta} \cdot \mathbf{f}(r, (i, j), (m, n), \mathbf{w}), \quad [17.12]$$

8957 with $\boldsymbol{\theta}$ representing a vector of weights, and $\mathbf{f}(\cdot)$ a vector of features. The pattern-based
 8958 methods described in § 17.2.1 suggest several features:

- 8959 • Local features of $\mathbf{w}_{i+1:j}$ and $\mathbf{w}_{m+1:n}$, including: the strings themselves; whether they
 8960 are recognized as entities, and if so, which type; whether the strings are present in a
 8961 **gazetteer** of entity names; each string's syntactic **head** (§ 9.2.2).
- 8962 • Features of the span between the two arguments, $\mathbf{w}_{j+1:m}$ or $\mathbf{w}_{n+1:i}$ (depending on
 8963 which argument appears first): the length of the span; the specific words that appear
 8964 in the span, either as a literal sequence or a bag-of-words; the wordnet synsets (§ 4.2)
 8965 that appear in the span between the arguments.
- 8966 • Features of the syntactic relationship between the two arguments, typically the **de-**
 8967 **pendency path** between the arguments (§ 13.2.1). Example dependency paths are
 8968 shown in Table 17.2.

8969 **17.2.2.2 Kernels**

8970 Suppose that the first line of Table 17.2 is a labeled example, and the remaining lines are
 8971 instances to be classified. A feature-based approach would have to decompose the depen-
 8972 dency paths into features that capture individual edges, with or without their labels, and
 8973 then learn weights for each of these features: for example, the second line contains identi-
 8974 cal dependencies, but different arguments; the third line contains a different inflection of
 8975 the word *travel*; the fourth and fifth lines each contain an additional edge on the depen-
 8976 dency path; and the sixth example uses an entirely different path. Rather than attempting
 8977 to create local features that capture all of the ways in which these dependencies paths

1. <i>George Bush traveled to France</i>	<i>George Bush</i> \leftarrow <i>traveled</i> \rightarrow <i>France</i> NSUBJ OBL
2. <i>Ahab traveled to Nantucket</i>	<i>Ahab</i> \leftarrow <i>traveled</i> \rightarrow <i>Nantucket</i> NSUBJ OBL
3. <i>George Bush will travel to France</i>	<i>George Bush</i> \leftarrow <i>travel</i> \rightarrow <i>France</i> NSUBJ OBL
4. <i>George Bush wants to travel to France</i>	<i>George Bush</i> \leftarrow <i>wants</i> \rightarrow <i>travel</i> \rightarrow <i>France</i> NSUBJ XCOMP OBL
5. <i>Ahab traveled to a city in France</i>	<i>Ahab</i> \leftarrow <i>traveled</i> \rightarrow <i>city</i> \rightarrow <i>France</i> NSUBJ OBL NMOD
6. <i>We await Ahab's visit to France</i>	<i>Ahab</i> \leftarrow <i>visit</i> \rightarrow <i>France</i> NMOD:POSS NMOD

Table 17.2: Candidates instances for the PHYSICAL.LOCATED relation, and their dependency paths

8978 are similar and different, we can instead define a similarity function κ , which computes a
 8979 score for any pair of instances, $\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}_+$. The score for any pair of instances (i, j)
 8980 is $\kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \geq 0$, with $\kappa(i, j)$ being large when instances $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ are similar. If the
 8981 function κ obeys a few key properties it is a valid **kernel function**.⁴

Given a valid kernel function, we can build a non-linear classifier without explicitly defining a feature vector or neural network architecture. For a binary classification problem $y \in \{-1, 1\}$, we have the decision function,

$$\hat{y} = \text{Sign}(b + \sum_{i=1}^N y^{(i)} \alpha^{(i)} \kappa(\mathbf{x}^{(i)}, \mathbf{x})) \quad [17.13]$$

8982 where b and $\{\alpha^{(i)}\}_{i=1}^N$ are parameters that must be learned from the training set, under
 8983 the constraint $\forall_i, \alpha^{(i)} \geq 0$. Intuitively, each α_i specifies the importance of the instance $\mathbf{x}^{(i)}$
 8984 towards the classification rule. Kernel-based classification can be viewed as a weighted
 8985 form of the **nearest-neighbor** classifier (Hastie et al., 2009), in which test instances are
 8986 assigned the most common label among their near neighbors in the training set. This
 8987 results in a non-linear classification boundary. The parameters are typically learned from
 8988 a margin-based objective (see § 2.3), leading to the **kernel support vector machine**. To
 8989 generalize to multi-class classification, we can train separate binary classifiers for each
 8990 label (sometimes called **one-versus-all**), or train binary classifiers for each pair of possible
 8991 labels (**one-versus-one**).

8992 Dependency kernels are particularly effective for relation extraction, due to their ability
 8993 to capture syntactic properties of the path between the two candidate arguments. One
 8994 class of dependency tree kernels is defined recursively, with the score for a pair of trees

⁴The **Gram matrix** \mathbf{K} arises from computing the kernel function between all pairs in a set of instances. For a valid kernel, the Gram matrix must be symmetric ($\mathbf{K} = \mathbf{K}^\top$) and positive semi-definite ($\forall \mathbf{a}, \mathbf{a}^\top \mathbf{K} \mathbf{a} \geq 0$). For more on kernel-based classification, see chapter 14 of Murphy (2012).

equal to the similarity of the root nodes and the sum of similarities of matched pairs of child subtrees (Zelenko et al., 2003; Culotta and Sorensen, 2004). Alternatively, Bunescu and Mooney (2005) define a kernel function over sequences of unlabeled dependency edges, in which the score is computed as a product of scores for each pair of words in the sequence: identical words receive a high score, words that share a synset or part-of-speech receive a small non-zero score (e.g., *travel* / *visit*), and unrelated words receive a score of zero.

17.2.2.3 Neural relation extraction

Convolutional neural networks were an early neural architecture for relation extraction (Zeng et al., 2014; dos Santos et al., 2015). For the sentence (w_1, w_2, \dots, w_M) , obtain a matrix of word embeddings \mathbf{X} , where $x_m \in \mathbb{R}^K$ is the embedding of w_m . Now, suppose the candidate arguments appear at positions a_1 and a_2 ; then for each word in the sentence, its position with respect to each argument is $m - a_1$ and $m - a_2$. (Following Zeng et al. (2014), this is a restricted version of the relation extraction task in which the arguments are single tokens.) To capture any information conveyed by these positions, the word embeddings are concatenated with embeddings of the positional offsets, $x_{m-a_1}^{(p)}$ and $x_{m-a_2}^{(p)}$. The complete base representation of the sentence is,

$$\mathbf{X}(a_1, a_2) = \begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_M \\ \mathbf{x}_{1-a_1}^{(p)} & \mathbf{x}_{2-a_1}^{(p)} & \cdots & \mathbf{x}_{M-a_1}^{(p)} \\ \mathbf{x}_{1-a_2}^{(p)} & \mathbf{x}_{2-a_2}^{(p)} & \cdots & \mathbf{x}_{M-a_2}^{(p)} \end{pmatrix}, \quad [17.14]$$

where each column is a vertical concatenation of a word embedding, represented by the column vector \mathbf{x}_m , and two positional embeddings, specifying the position with respect to a_1 and a_2 . The matrix $\mathbf{X}(a_1, a_2)$ is then taken as input to a convolutional layer (see § 3.4), and max-pooling is applied to obtain a vector. The final scoring function is then,

$$\Psi(r, i, j, \mathbf{X}) = \theta_r \cdot \text{MaxPool}(\text{ConvNet}(\mathbf{X}(i, j); \phi)), \quad [17.15]$$

where ϕ defines the parameters of the convolutional operator, and the θ_r defines a set of weights for relation r . The model can be trained using a margin objective,

$$\hat{r} = \underset{r}{\operatorname{argmax}} \Psi(r, i, j, \mathbf{X}) \quad [17.16]$$

$$\ell = (1 + \psi(\hat{r}, i, j, \mathbf{X}) - \psi(r, i, j, \mathbf{X}))_+. \quad [17.17]$$

Recurrent neural networks have also been applied to relation extraction, using a network such as an bidirectional LSTM to encode the words or dependency path between the two arguments. Xu et al. (2015) segment each dependency path into left and right subpaths: the path $George \xleftarrow[\text{NSUBJ}]{} Bush \xrightarrow[\text{XCOMP}]{} wants \xrightarrow[\text{OBL}]{} travel \xrightarrow{} France$ is segmented into the subpaths,

9021 (17.10) *George Bush* $\xleftarrow{\text{NSUBJ}}$ *wants*

9022 (17.11) *wants* $\xrightarrow{\text{XCOMP}}$ *travel* $\xrightarrow{\text{OBL}}$ *France*.

Xu et al. (2015) then run recurrent networks from the arguments to the root word (in this case, *wants*), obtaining the final representation by max pooling across all the recurrent states along each path. This process can be applied across separate “channels”, in which the inputs consist of embeddings for the words, parts-of-speech, dependency relations, and WordNet hypernyms. To define the model formally, let $s(m)$ define the successor of word m in either the left or right subpath (in a dependency path, each word can have a successor in at most one subpath). Let $x_m^{(c)}$ indicate the embedding of word (or relation) m in channel c , and let $\overleftarrow{h}_m^{(c)}$ and $\overrightarrow{h}_m^{(c)}$ indicate the associated recurrent states in the left and right subtrees respectively. Then the complete model is specified as follows,

$$h_{s(m)}^{(c)} = \text{RNN}(x_{s(m)}^{(c)}, h_m^{(c)}) \quad [17.18]$$

$$z^{(c)} = \text{MaxPool}(\overleftarrow{h}_i^{(c)}, \overleftarrow{h}_{s(i)}^{(c)}, \dots, \overleftarrow{h}_{\text{root}}^{(c)}, \overrightarrow{h}_j^{(c)}, \overrightarrow{h}_{s(j)}^{(c)}, \dots, \overrightarrow{h}_{\text{root}}^{(c)}) \quad [17.19]$$

$$\Psi(r, i, j) = \theta \cdot [z^{(\text{word})}; z^{(\text{POS})}; z^{(\text{dependency})}; z^{(\text{hypernym})}] \quad [17.20]$$

9023 Note that z is computed by applying max-pooling to the *matrix* of horizontally concatenated vectors h , while Ψ is computed from the *vector* of vertically concatenated vectors 9024 z . Xu et al. (2015) pass the score Ψ through a **softmax** layer to obtain a probability 9025 $p(r | i, j, w)$, and train the model by regularized **cross-entropy**. Miwa and Bansal (2016) 9026 show that a related model can solve the more challenging “end-to-end” relation extraction 9027 task, in which the model must simultaneously detect entities and then extract their 9028 relations.

9030 17.2.3 Knowledge base population

9031 In many applications, what matters is not what fraction of sentences are analyzed correctly, but how much accurate knowledge can be extracted. **Knowledge base population** 9032 (**KBP**) refers to the task of filling in Wikipedia-style infoboxes, as shown in Figure 17.1a. 9033 Knowledge base population can be decomposed into two subtasks: **entity linking** (described in § 17.1), and **slot filling** (Ji and Grishman, 2011). Slot filling has two key differences from the formulation of relation extraction presented above: the relations hold 9034 between entities rather than spans of text, and the performance is evaluated at the *type level* (on entity pairs), rather than on the *token level* (on individual sentences).

9035 From a practical standpoint, there are three other important differences between slot 9036 filling and per-sentence relation extraction.

- KBP tasks are often formulated from the perspective of identifying attributes of a few “query” entities. As a result, these systems often start with an **information retrieval** phase, in which relevant passages of text are obtained by search.
- For many entity pairs, there will be multiple passages of text that provide evidence. Slot filling systems must aggregate this evidence to predict a single relation type (or set of relations).
- Labeled data is usually available in the form of pairs of related entities, rather than annotated passages of text. Training from such type-level annotations is a challenge: two entities may be linked by several relations, or they may appear together in a passage of text that nonetheless does not describe their relation to each other.

Information retrieval is beyond the scope of this text (see Manning et al., 2008). The remainder of this section describes approaches to information fusion and learning from type-level annotations.

17.2.3.1 Information fusion

In knowledge base population, there will often be multiple pieces of evidence for (and sometimes against) a single relation. For example, a search for the entity MAYNARD JACKSON, JR. may return several passages that reference the entity ATLANTA:⁵

- (17.12) Elected mayor of Atlanta in 1973, **Maynard Jackson** was the first African American to serve as mayor of a major southern city.
- (17.13) **Atlanta's** airport will be renamed to honor **Maynard Jackson**, the city's first Black mayor .
- (17.14) Born in Dallas, Texas in 1938, **Maynard Holbrook Jackson, Jr.** moved to Atlanta when he was 8.
- (17.15) **Maynard Jackson** has gone from one of the worst high schools in **Atlanta** to one of the best.

The first and second examples provide evidence for the relation **MAYOR** holding between the entities **ATLANTA** and **MAYNARD JACKSON, JR.**. The third example provides evidence for a different relation between these same entities, **LIVED-IN**. The fourth example poses an entity linking problem, referring to **MAYNARD JACKSON HIGH SCHOOL**. Knowledge base population requires aggregating this sort of textual evidence, and predicting the relations that are most likely to hold.

⁵First three examples from: <http://www.georgiaencyclopedia.org/articles/government-politics/maynard-jackson-1938-2003>; JET magazine, November 10, 2003; www.todayingeorgiahistory.org/content/maynard-jackson-elected

9072 One approach is to run a single-document relation extraction system (using the tech-
 9073 niques described in § 17.2.2), and then aggregate the results (Li et al., 2011). Relations
 9074 that are detected with high confidence in multiple documents are more likely to be valid,
 9075 motivating the heuristic,

$$\psi(r, e_1, e_2) = \sum_{i=1}^N (\text{p}(r(e_1, e_2) | \mathbf{w}^{(i)}))^\alpha, \quad [17.21]$$

9076 where $\text{p}(r(e_1, e_2) | \mathbf{w}^{(i)})$ is the probability of relation r between entities e_1 and e_2 condi-
 9077 tioned on the text $\mathbf{w}^{(i)}$, and $\alpha \gg 1$ is a tunable hyperparameter. Using this heuristic, it is
 9078 possible to rank all candidate relations, and trace out a **precision-recall curve** as more rel-
 9079ations are extracted.⁶ Alternatively, features can be aggregated across multiple passages
 9080 of text, feeding a single type-level relation extraction system (Wolfe et al., 2017).

9081 Precision can be improved by introducing constraints across multiple relations. For
 9082 example, if we are certain of the relation $\text{PARENT}(e_1, e_2)$, then it cannot also be the case
 9083 that $\text{PARENT}(e_2, e_1)$. Integer linear programming makes it possible to incorporate such
 9084 constraints into a global optimization (Li et al., 2011). Other pairs of relations have pos-
 9085itive correlations, such $\text{MAYOR}(e_1, e_2)$ and $\text{LIVED-IN}(e_1, e_2)$. Compatibility across relation
 9086 types can be incorporated into probabilistic graphical models (e.g., Riedel et al., 2010).

9087 17.2.3.2 Distant supervision

9088 Relation extraction is “annotation hungry,” because each relation requires its own la-
 9089 beled data. Rather than relying on annotations of individual documents, it would be
 9090 preferable to use existing knowledge resources — such as the many facts that are al-
 9091 ready captured in knowledge bases like DBpedia. However such annotations raise the
 9092 inverse of the information fusion problem considered above: the existence of the relation
 9093 $\text{MAYOR}(\text{MAYNARD JACKSON JR., ATLANTA})$ provides only **distant supervision** for the
 9094 example texts in which this entity pair is mentioned.

9095 One approach is to treat the entity pair as the instance, rather than the text itself (Mintz
 9096 et al., 2009). Features are then aggregated across all sentences in which both entities are
 9097 mentioned, and labels correspond to the relation (if any) between the entities in a knowl-
 9098 edge base, such as FreeBase. Negative instances are constructed from entity pairs that are
 9099 not related in the knowledge base. In some cases, two entities are related, but the knowl-
 9100 edge base is missing the relation; however, because the number of possible entity pairs is
 9101 huge, these missing relations are presumed to be relatively rare. This approach is shown
 9102 in Figure 17.2.

⁶The precision-recall curve is similar to the ROC curve shown in Figure 4.4, but it includes the precision $\frac{\text{TP}}{\text{TP} + \text{FP}}$ rather than the false positive rate $\frac{\text{FP}}{\text{FP} + \text{TN}}$.

- **Label** : MAYOR(ATLANTA, MAYNARD JACKSON)
 - Elected mayor of **Atlanta** in 1973, **Maynard Jackson** ...
 - **Atlanta**'s airport will be renamed to honor **Maynard Jackson**, the city's first Black mayor
 - Born in Dallas, Texas in 1938, **Maynard Holbrook Jackson, Jr.** moved to **Atlanta** when he was 8.
- **Label** : MAYOR(NEW YORK, FIORELLO LA GUARDIA)
 - **Fiorello La Guardia** was Mayor of **New York** for three terms ...
 - **Fiorello La Guardia**, then serving on the **New York** City Board of Aldermen...
- **Label** : BORN-IN(DALLAS, MAYNARD JACKSON)
 - Born in **Dallas**, Texas in 1938, **Maynard Holbrook Jackson, Jr.** moved to Atlanta when he was 8.
 - **Maynard Jackson** was raised in **Dallas** ...
- **Label** : NIL(NEW YORK, MAYNARD JACKSON)
 - **Jackson** married Valerie Richardson, whom he had met in **New York**...
 - **Jackson** was a member of the Georgia and **New York** bars ...

Figure 17.2: Four training instances for relation classification using **distant supervision** Mintz et al. (2009). The first two instances are positive for the MAYOR relation, and the third instance is positive for the BORN-IN relation. The fourth instance is a negative example, constructed from a pair of entities (NEW YORK, MAYNARD JACKSON) that do not appear in any Freebase relation. Each instance's features are computed by aggregating across all sentences in which the two entities are mentioned.

9103 In **multiple instance learning**, labels are assigned to *sets* of instances, of which only
 9104 an unknown subset are actually relevant (Dietterich et al., 1997; Maron and Lozano-Pérez,
 9105 1998). This formalizes the framework of distant supervision: the relation $\text{REL}(A, B)$ acts
 9106 as a label for the entire set of sentences mentioning entities A and B, even when only a
 9107 subset of these sentences actually describes the relation. One approach to multi-instance
 9108 learning is to introduce a binary **latent variable** for each sentence, indicating whether the
 9109 sentence expresses the labeled relation (Riedel et al., 2010). A variety of inference tech-
 9110 niques have been employed for this probabilistic model of relation extraction: Surdeanu
 9111 et al. (2012) use expectation maximization, Riedel et al. (2010) use sampling, and Hoff-
 9112 mann et al. (2011) use a custom graph-based algorithm. Expectation maximization and
 9113 sampling are surveyed in chapter 5, and are covered in more detail by Murphy (2012);
 9114 graph-based methods are surveyed by Mihalcea and Radev (2011).

Task	Relation ontology	Supervision
PropBank semantic role labeling	VerbNet	sentence
FrameNet semantic role labeling	FrameNet	sentence
Relation extraction	ACE, TAC, SemEval, etc	sentence
Slot filling	ACE, TAC, SemEval, etc	relation
Open Information Extraction	open	seed relations or patterns

Table 17.3: Various relation extraction tasks and their properties. VerbNet and FrameNet are described in chapter 13. ACE (Linguistic Data Consortium, 2005), TAC (McNamee and Dang, 2009), and SemEval (Hendrickx et al., 2009) refer to shared tasks, each of which involves an ontology of relation types.

9115 17.2.4 Open information extraction

9116 In classical relation extraction, the set of relations is defined in advance, using a **schema**.
 9117 The relation for any pair of entities can then be predicted using multi-class classification.
 9118 In **open information extraction** (OpenIE), a relation can be any triple of text. The example
 9119 sentence (17.12) instantiates several “relations” of this sort:

- 9120 • (*mayor of, Maynard Jackson, Atlanta*),
 9121 • (*elected, Maynard Jackson, mayor of Atlanta*),
 9122 • (*elected in, Maynard Jackson, 1973*),

9123 and so on. Extracting such tuples can be viewed as a lightweight version of **semantic role**
 9124 **labeling** (chapter 13), with only two argument types: first slot and second slot. The task is
 9125 generally evaluated on the relation level, rather than on the level of sentences: precision is
 9126 measured by the number of extracted relations that are accurate, and recall is measured by
 9127 the number of true relations that were successfully extracted. OpenIE systems are trained
 9128 from distant supervision or bootstrapping, rather than from labeled sentences.

9129 An early example is the TextRunner system (Banko et al., 2007), which identifies re-
 9130 lations with a set of handcrafted syntactic rules. The examples that are acquired from the
 9131 handcrafted rules are then used to train a classification model that uses part-of-speech pat-
 9132 terns as features. Finally, the relations that are extracted by the classifier are aggregated,
 9133 removing redundant relations and computing the number of times that each relation is
 9134 mentioned in the corpus. TextRunner was the first in a series of systems that performed
 9135 increasingly accurate open relation extraction by incorporating more precise linguistic fea-
 9136 tures (Etzioni et al., 2011), distant supervision from Wikipedia infoboxes (Wu and Weld,
 9137 2010), and better learning algorithms (Zhu et al., 2009).

9138 17.3 Events

9139 Relations link pairs of entities, but many real-world situations involve more than two en-
9140 tities. Consider again the example sentence (17.12), which describes the **event** of an elec-
9141 tion, with four properties: the office (MAYOR), the district (ATLANTA), the date (1973), and
9142 the person elected (MAYNARD JACKSON, JR.). In **event detection**, a schema is provided
9143 for each event type (e.g., an election, a terrorist attack, or a chemical reaction), indicating
9144 all the possible properties of the event. The system is then required to fill in as many of
9145 these properties as possible (Doddington et al., 2004).

9146 Event detection systems generally involve a retrieval component (finding relevant
9147 documents and passages of text) and an extraction component (determining the proper-
9148 ties of the event based on the retrieved texts). Early approaches focused on finite-state pat-
9149 terns for identify event properties (Hobbs et al., 1997); such patterns can be automatically
9150 induced by searching for patterns that are especially likely to appear in documents that
9151 match the event query (Riloff, 1996). Contemporary approaches employ techniques that
9152 are similar to FrameNet semantic role labeling (§ 13.2), such as structured prediction over
9153 local and global features (Li et al., 2013) and bidirectional recurrent neural networks (Feng
9154 et al., 2016). These methods detect whether an event is described in a sentence, and if so,
9155 what are its properties.

9156 **Event coreference** Because multiple sentences may describe unique properties of a sin-
9157 gle event, **event coreference** is required to link event mentions across a single passage
9158 of text, or between passages (Humphreys et al., 1997). Bejan and Harabagiu (2014) de-
9159 fine event coreference as the task of identifying event mentions that share the same event
9160 participants (i.e., the slot-filling entities) and the same event properties (e.g., the time and
9161 location), within or across documents. Event coreference resolution can be performed us-
9162 ing supervised learning techniques in a similar way to entity coreference, as described
9163 in chapter 15: move left-to-right through the document, and use a classifier to decide
9164 whether to link each event reference to an existing cluster of coreferent events, or to cre-
9165 ate a new cluster (Ahn, 2006). Each clustering decision is based on the compatibility of
9166 features describing the participants and properties of the event. Due to the difficulty of
9167 annotating large amounts of data for entity coreference, unsupervised approaches are es-
9168 pecially desirable (Chen and Ji, 2009; Bejan and Harabagiu, 2014).

9169 **Relations between events** Just as entities are related to other entities, events may be
9170 related to other events: for example, the event of winning an election both *precedes* and
9171 *causes* the event of serving as mayor; moving to Atlanta *precedes* and *enables* the event of
9172 becoming mayor of Atlanta; moving from Dallas to Atlanta *prevents* the event of later be-
9173 coming mayor of Dallas. As these examples show, events may be related both temporally
9174 and causally. The **TimeML** annotation scheme specifies a set of six temporal relations

	Positive (+)	Negative (-)	Underspecified (u)
Certain (CT)	Fact: CT+	Counterfact: CT-	Certain, but unknown: CTU
Probable (PR)	Probable: PR+	Not probable: PR-	(NA)
Possible (PS)	Possible: PS+	Not possible: PS-	(NA)
Underspecified (U)	(NA)	(NA)	Unknown or uncommitted: UU

Table 17.4: Table of factuality values from the FactBank corpus (Saurí and Pustejovsky, 2009). The entry (NA) indicates that this combination is not annotated.

9175 between events (Pustejovsky et al., 2005), derived in part from **interval algebra** (Allen,
9176 1984). The TimeBank corpus provides TimeML annotations for 186 documents (Pustejovsky
9177 et al., 2003). Methods for detecting these temporal relations combine supervised
9178 machine learning with temporal constraints, such as transitivity (e.g. Mani et al., 2006;
9179 Chambers and Jurafsky, 2008).

9180 More recent annotation schemes and datasets combine temporal and causal relations (Mirza
9181 et al., 2014; Dunietz et al., 2017): for example, the CaTeRS dataset includes annotations of
9182 320 five-sentence short stories (Mostafazadeh et al., 2016). Abstracting still further, **processes**
9183 are networks of causal relations between multiple events. A small dataset of bi-
9184 ological processes is annotated in the ProcessBank dataset (Berant et al., 2014), with the
9185 goal of supporting automatic question answering on scientific textbooks.

9186 17.4 Hedges, denials, and hypotheticals

9187 The methods described thus far apply to **propositions** about the way things are in the
9188 real world. But natural language can also describe events and relations that are likely or
9189 unlikely, possible or impossible, desired or feared. The following examples hint at the
9190 scope of the problem (Prabhakaran et al., 2010):

- 9191 (17.16) GM will lay off workers.
- 9192 (17.17) A spokesman for GM said GM will lay off workers.
- 9193 (17.18) GM may lay off workers.
- 9194 (17.19) The politician claimed that GM will lay off workers.
- 9195 (17.20) Some wish GM would lay off workers.
- 9196 (17.21) Will GM lay off workers?
- 9197 (17.22) Many wonder whether GM will lay off workers.

9198 Accurate information extraction requires handling these **extra-propositional** aspects
 9199 of meaning, which are sometimes summarized under the terms **modality** and **negation**.⁷
 9200 Modality refers to expressions of the speaker's attitude towards her own statements, in-
 9201 cluding "degree of certainty, reliability, subjectivity, sources of information, and perspec-
 9202 tive" (Morante and Sporleder, 2012). Various systematizations of modality have been
 9203 proposed (e.g., Palmer, 2001), including categories such as future, interrogative, imper-
 9204 ative, conditional, and subjective. Information extraction is particularly concerned with
 9205 negation and certainty. For example, Saurí and Pustejovsky (2009) link negation with
 9206 a modal calculus of certainty, likelihood, and possibility, creating the two-dimensional
 9207 schema shown in Table 17.4. This is the basis for the FactBank corpus, with annotations
 9208 of the **factuality** of all sentences in 208 documents of news text.

9209 A related concept is **hedging**, in which speakers limit their commitment to a proposi-
 9210 tion (Lakoff, 1973):

- 9211 (17.23) These results **suggest** that expression of c-jun, jun B and jun D genes **might** be in-
 9212 volved in terminal granulocyte differentiation... (Morante and Daelemans, 2009)
- 9213 (17.24) A whale is **technically** a mammal (Lakoff, 1973)

9214 In the first example, the hedges *suggest* and *might* communicate uncertainty; in the second
 9215 example, there is no uncertainty, but the hedge *technically* indicates that the evidence for
 9216 the proposition will not fully meet the reader's expectations. Hedging has been studied
 9217 extensively in scientific texts (Medlock and Briscoe, 2007; Morante and Daelemans, 2009),
 9218 where the goal of large-scale extraction of scientific facts is obstructed by hedges and spec-
 9219 ulation. Still another related aspect of modality is **evidentiality**, in which speakers mark
 9220 the source of their information. In many languages, it is obligatory to mark evidentiality
 9221 through affixes or particles (Aikhenvald, 2004); while evidentiality is not grammaticalized
 9222 in English, authors are expected to express this information in contexts such as journal-
 9223 ism (Kovach and Rosenstiel, 2014) and Wikipedia.⁸

9224 Methods for handling negation and modality generally include two phases:

- 9225 1. detecting negated or uncertain events;
- 9226 2. identifying the scope and focus of the negation or modal operator.

⁷The classification of negation as extra-propositional is controversial: Packard et al. (2014) argue that negation is a "core part of compositionally constructed logical-form representations." Negation is an element of the semantic parsing tasks discussed in chapter 12 and chapter 13 — for example, negation markers are treated as adjuncts in PropBank semantic role labeling. However, many of the relation extraction methods mentioned in this chapter do not handle negation directly. A further consideration is that negation interacts closely with aspects of modality that are generally not considered in propositional semantics, such as certainty and subjectivity.

⁸<https://en.wikipedia.org/wiki/Wikipedia:Verifiability>

9227 A considerable body of work on negation has employed rule-based techniques such as
 9228 regular expressions (Chapman et al., 2001) to detect negated events. Such techniques
 9229 match lexical cues (e.g., *Norwood was **not** elected Mayor*), while avoiding “double nega-
 9230 tives” (e.g., *surely all this is **not without** meaning*). More recent approaches employ classi-
 9231 fiers over lexical and syntactic features (Uzuner et al., 2009) and sequence labeling (Prab-
 9232 hakaran et al., 2010).

9233 The tasks of scope and focus resolution are more fine grained, as shown in the example
 9234 from Morante and Sporleder (2012):

- 9235 (17.25) [After his habit he said] **nothing**, and after mine I asked no questions.
 9236 After his habit he said nothing, and [after mine I asked] **no** [questions].

9237 In this sentence, there are two negation cues (*nothing* and *no*). Each negates an event,
 9238 indicated by the underlined verbs *said* and *asked* (this is the focus of negation), and each
 9239 occurs within a scope: *after his habit he said* and *after mine I asked* ____ *questions*. These tasks
 9240 are typically formalized as sequence labeling problems, with each word token labeled
 9241 as beginning, inside, or outside of a cue, focus, or scope span (see § 8.3). Conventional
 9242 sequence labeling approaches can then be applied, using surface features as well as syn-
 9243 tax (Velldal et al., 2012) and semantic analysis (Packard et al., 2014). Labeled datasets
 9244 include the BioScope corpus of biomedical texts (Vincze et al., 2008) and a shared task
 9245 dataset of detective stories by Arthur Conan Doyle (Morante and Blanco, 2012).

9246 17.5 Question answering and machine reading

9247 The victory of the Watson question-answering system against three top human players on
 9248 the game show *Jeopardy!* was a landmark moment for natural language processing (Fer-
 9249 rucci et al., 2010). Game show questions are usually answered by **factoids**: entity names
 9250 and short phrases.⁹ The task of factoid question answering is therefore closely related to
 9251 information extraction, with the additional problem of accurately parsing the question.

9252 17.5.1 Formal semantics

9253 Semantic parsing is an effective method for question-answering in restricted domains
 9254 such as questions about geography and airline reservations (Zettlemoyer and Collins,
 9255 2005), and has also been applied in “open-domain” settings such as question answering
 9256 on Freebase (Berant et al., 2013) and biomedical research abstracts (Poon and Domingos,
 9257 2009). One approach is to convert the question into a lambda calculus expression that

⁹The broader landscape of question answering includes “why” questions (*Why did Ahab continue to pursue the white whale?*), “how questions” (*How did Queequeg die?*), and requests for summaries (*What was Ishmael’s attitude towards organized religion?*). For more, see Hirschman and Gaizauskas (2001).

9258 returns a boolean value: for example, the question *who is the mayor of the capital of Georgia?*
 9259 would be converted to,

$$\lambda x. \exists y \text{ CAPITAL(GEORGIA, } y) \wedge \text{MAYOR}(y, x). \quad [17.22]$$

9260 This lambda expression can then be used to query an existing knowledge base, returning
 9261 “true” for all entities that satisfy it.

9262 17.5.2 Machine reading

9263 Recent work has focused on answering questions about specific textual passages, similar
 9264 to the reading comprehension examinations for young students (Hirschman et al., 1999).
 9265 This task has come to be known as **machine reading**.

9266 17.5.2.1 Datasets

9267 The machine reading problem can be formulated in a number of different ways. The most
 9268 important distinction is what form the answer should take.

- 9269 • **Multiple-choice question answering**, as in the MCTest dataset of stories (Richard-
 9270 son et al., 2013) and the New York Regents Science Exams (Clark, 2015). In MCTest,
 9271 the answer is deducible from the text alone, while in the science exams, the system
 9272 must make inferences using an existing model of the underlying scientific phenom-
 9273 ena. Here is an example from MCTest:

9274 (17.26) James the turtle was always getting into trouble. Sometimes he'd reach into
 9275 the freezer and empty out all the food ...

9276 Q: What is the name of the trouble making turtle?
 9277 (a) Fries

9278 (b) Pudding
 9279 (c) James
 9280 (d) Jane

- 9281 • **Cloze-style “fill in the blank” questions**, as in the CNN/Daily Mail comprehension
 9282 task (Hermann et al., 2015), the Children’s Book Test (Hill et al., 2016), and the Who-
 9283 did-What dataset (Onishi et al., 2016). In these tasks, the system must guess which
 9284 word or entity completes a sentence, based on reading a passage of text. Here is an
 9285 example from Who-did-What:

9286 (17.27) Q: Tottenham manager Juande Ramos has hinted he will allow ____ to leave
 9287 if the Bulgaria striker makes it clear he is unhappy. (Onishi et al., 2016)

The query sentence may be selected either from the story itself, or from an external summary. In either case, datasets can be created automatically by processing large quantities existing documents. An additional constraint is that that missing element from the cloze must appear in the main passage of text: for example, in Who-did-What, the candidates include all entities mentioned in the main passage. In the CNN/Daily Mail dataset, each entity name is replaced by a unique identifier, e.g., ENTITY37. This ensures that correct answers can only be obtained by accurately reading the text, and not from external knowledge about the entities.

- **Extractive** question answering, in which the answer is drawn from the original text. In WikiQA, answers are sentences (Yang et al., 2015). In the Stanford Question Answering Dataset (SQuAD), answers are words or short phrases (Rajpurkar et al., 2016):

(17.28) In metereology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity.

Q: What causes precipitation to fall? A: gravity

In both WikiQA and SQuAD, the original texts are Wikipedia articles, and the questions are generated by crowdworkers.

17.5.2.2 Methods

A baseline method is to search the text for sentences or short passages that overlap with both the query and the candidate answer (Richardson et al., 2013). In example (17.26), this baseline would select the correct answer, since *James* appears in a sentence that includes the query terms *trouble* and *turtle*.

This baseline can be implemented as a neural architecture, using an **attention mechanism** (see § 18.3.1), which scores the similarity of the query to each part of the source text (Chen et al., 2016). The first step is to encode the passage $w^{(p)}$ and the query $w^{(q)}$, using two bidirectional LSTMs (§ 7.6).

$$\mathbf{h}^{(q)} = \text{BiLSTM}(\mathbf{w}^{(q)}; \Theta^{(q)}) \quad [17.23]$$

$$\mathbf{h}^{(p)} = \text{BiLSTM}(\mathbf{w}^{(p)}; \Theta^{(p)}). \quad [17.24]$$

The query is represented by vertically concatenating the final states of the left-to-right and right-to-left passes:

$$\mathbf{u} = [\overrightarrow{\mathbf{h}}^{(q)}_{M_q}; \overleftarrow{\mathbf{h}}^{(q)}_0]. \quad [17.25]$$

The attention vector is computed as a softmax over a vector of bilinear products, and the expected representation is computed by summing over attention values,

$$\tilde{\alpha}_m = (\mathbf{u}^{(q)})^\top \mathbf{W}_a \mathbf{h}_m^{(p)} \quad [17.26]$$

$$\boldsymbol{\alpha} = \text{SoftMax}(\tilde{\boldsymbol{\alpha}}) \quad [17.27]$$

$$\mathbf{o} = \sum_{m=1}^M \alpha_m \mathbf{h}_m^{(p)}. \quad [17.28]$$

Each candidate answer c is represented by a vector \mathbf{x}_c . Assuming the candidate answers are spans from the original text, these vectors can be set equal to the corresponding element in $\mathbf{h}^{(p)}$. The score for each candidate answer a is computed by the inner product,

$$\hat{c} = \underset{c}{\operatorname{argmax}} \mathbf{o} \cdot \mathbf{x}_c. \quad [17.29]$$

9310 This architecture can be trained end-to-end from a loss based on the log-likelihood of the
 9311 correct answer. A number of related architectures have been proposed (e.g., Hermann
 9312 et al., 2015; Kadlec et al., 2016; Dhingra et al., 2017; Cui et al., 2017), and the relationships
 9313 between these methods are surveyed by Wang et al. (2017).

9314 Additional resources

9315 The field of information extraction is surveyed in course notes by Grishman (2012), and
 9316 more recently in a short survey paper (Grishman, 2015). Shen et al. (2015) survey the task
 9317 of entity linking, and Ji and Grishman (2011) survey work on knowledge base popula-
 9318 tion. This chapter’s discussion of non-propositional meaning was strongly influenced by
 9319 Morante and Sporleder (2012), who introduced a special issue of the journal *Computational
 9320 Linguistics* dedicated to recent work on modality and negation.

9321 Exercises

9322 1. Consider the following heuristic for entity linking:

- 9323 • Among all entities that have the same type as the mention (e.g., LOC, PER),
 9324 choose the one whose name has the lowest edit distance from the mention.
- 9325 • If more than one entity has the right type and the lowest edit distance from the
 9326 mention, choose the most popular one.
- 9327 • If no candidate entity has the right type, choose NIL.

Now suppose you have the following feature function:

$$f(y, \mathbf{x}) = [\text{edit-dist}(\text{name}(y), \mathbf{x}), \text{same-type}(y, \mathbf{x}), \text{popularity}(y), \delta(y = \text{NIL})]$$

Design a set of ranking weights θ that match the heuristic. You may assume that edit distance and popularity are always in the range [0, 100], and that the NIL entity has values of zero for all features except δ ($y = \text{NIL}$).

2. Now consider another heuristic:

- Among all candidate entities that have edit distance zero from the mention and the right type, choose the most popular one.
- If no entity has edit distance zero from the mention, choose the one with the right type that is most popular, regardless of edit distance.
- If no entity has the right type, choose NIL.

Using the same features and assumptions from the previous problem, prove that there is no set of weights that could implement this heuristic. Then show that the heuristic can be implemented by adding a single feature. Your new feature should consider only the edit distance.

3. * Consider the following formulation for collective entity linking, which rewards sets of entities that are all of the same type, where “types” can be elements of any set:

$$\psi_c(\mathbf{y}) = \begin{cases} \alpha & \text{all entities in } \mathbf{y} \text{ have the same type} \\ \beta & \text{more than half of the entities in } \mathbf{y} \text{ have the same type} \\ 0 & \text{otherwise.} \end{cases} \quad [17.30]$$

Show how to implement this model of collective entity linking in an **integer linear program**. You may want to review § 13.2.2.

To get started, here is an integer linear program for entity linking, without including the collective term ψ_c :

$$\begin{aligned} \max_{z_{i,y} \in \{0,1\}} \quad & \sum_{i=1}^N \sum_{y \in \mathcal{Y}(\mathbf{x}^{(i)})} s_{i,y} z_{i,y} \\ \text{s.t.} \quad & \sum_{y \in \mathcal{Y}(\mathbf{x}^{(i)})} z_{i,y} \leq 1 \quad \forall i \in \{1, 2, \dots, N\} \end{aligned}$$

where $z_{i,y} = 1$ if entity y is linked to mention i , and $s_{i,y}$ is a parameter that scores the quality of this individual ranking decision, e.g., $s_{i,y} = \theta \cdot f(y, \mathbf{x}^{(i)}, \mathbf{c}^{(i)})$.

To incorporate the collective linking score, you may assume parameters r ,

$$r_{y,\tau} = \begin{cases} 1, & \text{entity } y \text{ has type } \tau \\ 0, & \text{otherwise.} \end{cases} \quad [17.31]$$

Hint: You will need to define several auxiliary variables to optimize over.

- 9350 4. Run `nltk.corpus.download('reuters')` to download the Reuters corpus in
 9351 NLTK, and run `from nltk.corpus import reuters` to import it. The com-
 9352 mand `reuters.words()` returns an iterator over the tokens in the corpus.
 9353 a) Apply the pattern *_____, such as _____* to this corpus, obtaining candidates for the
 9354 IS-A relation, e.g. `IS-A(ROMANIA, COUNTRY)`. What are three pairs that this
 9355 method identifies correctly? What are three different pairs that it gets wrong?
 9356 b) Design a pattern for the PRESIDENT relation, e.g. `PRESIDENT(PHILIPPINES, CORAZON AQUINO)`.
 9357 In this case, you may want to augment your pattern matcher with the ability
 9358 to match multiple token wildcards, perhaps using case information to detect
 9359 proper names. Again, list three correct
 9360 c) Preprocess the Reuters data by running a named entity recognizer, replacing
 9361 tokens with named entity spans when applicable. Apply your PRESIDENT
 9362 matcher to this new data. Does the accuracy improve? Compare 20 randomly-
 9363 selected pairs from this pattern and the one you designed in the previous part.
- 9364 5. Represent the dependency path $\mathbf{x}^{(i)}$ as a sequence of words and dependency arcs
 9365 of length M_i , ignoring the endpoints of the path. In example 1 of Table 17.2, the
 9366 dependency path is,

$$\mathbf{x}^{(1)} = (\xleftarrow[\text{NSUBJ}]{}, \text{traveled}, \xrightarrow[\text{OBL}]{}) \quad [17.32]$$

9367 If $x_m^{(i)}$ is a word, then let $\text{pos}(x_m^{(i)})$ be its part-of-speech, using the tagset defined in
 9368 chapter 8.

We can define the following kernel function over pairs of dependency paths (Bunescu and Mooney, 2005):

$$\kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \begin{cases} 0, & M_i \neq M_j \\ \prod_{m=1}^{M_i} c(x_m^{(i)}, x_m^{(j)}), & M_i = M_j \end{cases}$$

$$c(x_m^{(i)}, x_m^{(j)}) = \begin{cases} 2, & x_m^{(i)} = x_m^{(j)} \\ 1, & x_m^{(i)} \text{ and } x_m^{(j)} \text{ are words and } \text{pos}(x_m^{(i)}) = \text{pos}(x_m^{(j)}) \\ 0, & \text{otherwise.} \end{cases}$$

9369 Using this kernel function, compute the kernel similarities of example 1 from Ta-
 9370 ble 17.2 with the other five examples.

- 9371 6. Continuing from the previous problem, suppose that the instances have the follow-
 9372 ing labels:

$$y_2 = 1, y_3 = -1, y_4 = -1, y_5 = 1, y_6 = 1 \quad [17.33]$$

Identify the conditions for α and b under which $\hat{y}_1 = 1$. Remember the constraint
 that $\alpha_i \geq 0$ for all i .

9373 Chapter 18

9374 Machine translation

9375 Machine translation (MT) is one of the “holy grail” problems in artificial intelligence,
9376 with the potential to transform society by facilitating communication between people
9377 anywhere in the world. As a result, MT has received significant attention and funding
9378 since the early 1950s. However, it has proved remarkably challenging, and while there
9379 has been substantial progress towards usable MT systems — especially for high-resource
9380 language pairs like English-French — we are still far from translation systems that match
9381 the nuance and depth of human translations.

9382 18.1 Machine translation as a task

9383 Machine translation can be formulated as an optimization problem:

$$\hat{\mathbf{w}}^{(t)} = \underset{\mathbf{w}^{(t)}}{\operatorname{argmax}} \Psi(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}), \quad [18.1]$$

9384 where $\mathbf{w}^{(s)}$ is a sentence in a **source** language, $\mathbf{w}^{(t)}$ is a sentence in the **target language**,
9385 and Ψ is a scoring function. As usual, this formalism requires two components: a decod-
9386 ing algorithm for computing $\hat{\mathbf{w}}^{(t)}$, and a learning algorithm for estimating the parameters
9387 of the scoring function Ψ .

9388 Decoding is difficult for machine translation because of the huge space of possible
9389 translations. We have faced large label spaces before: for example, in sequence labeling,
9390 the set of possible label sequences is exponential in the length of the input. In these cases,
9391 it was possible to search the space quickly by introducing locality assumptions: for ex-
9392 ample, that each tag depends only on its predecessor, or that each production depends
9393 only on its parent. In machine translation, no such locality assumptions seem possible:
9394 human translators reword, reorder, and rearrange words; they replace single words with
9395 multi-word phrases, and vice versa. This flexibility means that in even relatively simple

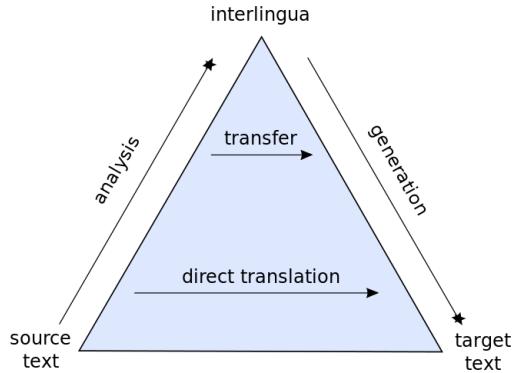


Figure 18.1: The Vauquois Pyramid http://commons.wikimedia.org/wiki/File:Direct_translation_and_transfer_translation_pyramind.svg

9396 translation models, decoding is NP-hard (Knight, 1999). Approaches for dealing with this
 9397 complexity are described in § 18.4.

Estimating translation models is difficult as well. Labeled translation data usually comes in the form parallel sentences, e.g.,

$$\begin{aligned} w^{(s)} &= A \text{ } Vinay \text{ } le \text{ } gusta \text{ } las \text{ } manzanas. \\ w^{(t)} &= Vinay \text{ likes apples.} \end{aligned}$$

9398 A useful feature function would note the translation pairs (*gusta, likes*), (*manzanas, apples*),
 9399 and even (*Vinay, Vinay*). But this word-to-word **alignment** is not given in the data. One
 9400 solution is to treat this alignment as a **latent variable**; this is the approach taken by clas-
 9401 sical **statistical machine translation** (SMT) systems, described in § 18.2. Another solution
 9402 is to model the relationship between $w^{(t)}$ and $w^{(s)}$ through a more complex and expres-
 9403 sive function; this is the approach taken by **neural machine translation** (NMT) systems,
 9404 described in § 18.3.

9405 The **Vauquois Pyramid** is a theory of how translation should be done. At the lowest
 9406 level, the translation system operates on individual words, but the horizontal distance
 9407 at this level is large, because languages express ideas differently. If we can move up the
 9408 triangle to syntactic structure, the distance for translation is reduced; we then need only
 9409 produce target-language text from the syntactic representation, which can be as simple
 9410 as reading off a tree. Further up the triangle lies semantics; translating between semantic
 9411 representations should be easier still, but mapping between semantics and surface text is
 9412 a difficult, unsolved problem. At the top of the triangle is **interlingua**, a semantic repre-
 9413 sentation that is so generic that it is identical across all human languages. Philosophers

	Adequate?	Fluent?
<i>To Vinay it like Python</i>	yes	no
<i>Vinay debugs memory leaks</i>	no	yes
<i>Vinay likes Python</i>	yes	yes

Table 18.1: Adequacy and fluency for translations of the Spanish sentence *A Vinay le gusta Python*.

debate whether such a thing as interlingua is really possible (Derrida, 1985). While the first-order logic representations discussed in chapter 12 might be considered to be language independent, it is built on an inventory of relations that is suspiciously similar to a subset of English words (Nirenburg and Wilks, 2001). Nonetheless, the idea of linking translation and semantic understanding may still be a promising path, if the resulting translations better preserve the meaning of the original text.

18.1.1 Evaluating translations

There are two main criteria for a translation, summarized in Table 18.1.

- **Adequacy:** The translation $w^{(t)}$ should adequately reflect the linguistic content of $w^{(s)}$. For example, if $w^{(s)} = A Vinay le gusta Python$, the gloss¹ $w^{(t)} = To Vinay it like Python$ is considered adequate because it contains all the relevant content. The output $w^{(t)} = Vinay debugs memory leaks$ is not adequate.
- **Fluency:** The translation $w^{(t)}$ should read like fluent text in the target language. By this criterion, the gloss $w^{(t)} = To Vinay it like Python$ will score poorly, and $w^{(t)} = Vinay debugs memory leaks$ will be preferred.

Automated evaluations of machine translations typically merge both of these criteria, by comparing the system translation with one or more **reference translations**, produced by professional human translators. The most popular quantitative metric is **BLEU** (bilingual evaluation understudy; Papineni et al., 2002), which is based on n -gram precision: what fraction of n -grams in the system translation appear in the reference? Specifically, for each n -gram length, the precision is defined as,

$$p_n = \frac{\text{number of } n\text{-grams appearing in both reference and hypothesis translations}}{\text{number of } n\text{-grams appearing in the hypothesis translation}}. \quad [18.2]$$

The n -gram precisions for three hypothesis translations are shown in Figure 18.2.

¹A gloss is a word-for-word translation.

	Translation	p_1	p_2	p_3	p_4	BP	BLEU
<i>Reference</i>	<i>Vinay likes programming in Python</i>						
<i>Sys1</i>	<i>To Vinay it like to program Python</i>	$\frac{2}{7}$	0	0	0	1	.21
<i>Sys2</i>	<i>Vinay likes Python</i>	$\frac{3}{3}$	$\frac{1}{2}$	0	0	.51	.33
<i>Sys3</i>	<i>Vinay likes programming in his pajamas</i>	$\frac{4}{6}$	$\frac{3}{5}$	$\frac{2}{4}$	$\frac{1}{3}$	1	.76

Figure 18.2: A reference translation and three system outputs. For each output, p_n indicates the precision at each n -gram, and BP indicates the brevity penalty.

9436 The BLEU score is then based on the average, $\exp \frac{1}{N} \sum_{n=1}^N \log p_n$. Two modifications
 9437 of Equation 18.2 are necessary: (1) to avoid computing $\log 0$, all precisions are smoothed
 9438 to ensure that they are positive; (2) each n -gram in the source can be used at most once,
 9439 so that *to to to to to* does not achieve $p_1 = 1$ against the reference *to be or not to be*.
 9440 Furthermore, precision-based metrics are biased in favor of short translations, which can
 9441 achieve high scores by minimizing the denominator in [18.2]. To avoid this issue, a **brevity**
 9442 **penalty** is applied to translations that are shorter than the reference. This penalty is indi-
 9443 cated as “BP” in Figure 18.2.

9444 Automated metrics like BLEU have been validated by correlation with human judg-
 9445 ments of translation quality. Nonetheless, it is not difficult to construct examples in which
 9446 the BLEU score is high, yet the translation is disfluent or carries a completely different
 9447 meaning from the original. To give just one example, consider the problem of translating
 9448 pronouns. Because pronouns refer to specific entities, a single incorrect pronoun can obliti-
 9449 onate the semantics of the original sentence. Existing state-of-the-art systems generally
 9450 do not attempt the reasoning necessary to correctly resolve pronominal anaphora (Hard-
 9451 meier, 2012). Despite the importance of pronouns for semantics, they have a marginal
 9452 impact on BLEU, which may help to explain why existing systems do not make a greater
 9453 effort to translate them correctly.

9454 **Fairness and bias** The problem of pronoun translation intersects with issues of fairness
 9455 and bias. In many languages, such as Turkish, the third person singular pronoun is gender
 9456 neutral. Today’s state-of-the-art systems produce the following Turkish-English transla-
 9457 tions (Caliskan et al., 2017):

- 9458 (18.1) *O bir doktor.*
 He is a doctor.
 9459 (18.2) *O bir hemşire.*
 She is a nurse.

9460 The same problem arises for other professions that have stereotypical genders, such as
9461 engineers, soldiers, and teachers, and for other languages that have gender-neutral pro-
9462 nouns. This bias was not directly programmed into the translation model; it arises from
9463 statistical tendencies in existing datasets. This highlights a general problem with data-
9464 driven approaches, which can perpetuate biases that negatively impact disadvantaged
9465 groups. Worse, machine learning can *amplify* biases in data (Bolukbasi et al., 2016): if a
9466 dataset has even a slight tendency towards men as doctors, the resulting translation model
9467 may produce translations in which doctors are always *he*, and nurses are always *she*.

9468 **Other metrics** A range of other automated metrics have been proposed for machine
9469 translation. One potential weakness of BLEU is that it only measures precision; METEOR
9470 is a weighted *F*-MEASURE, which is a combination of recall and precision (see § 4.4.1).
9471 **Translation Error Rate (TER)** computes the string **edit distance** (see § 9.1.4.1) between the
9472 reference and the hypothesis (Snover et al., 2006). For language pairs like English and
9473 Japanese, there are substantial differences in word order, and word order errors are not
9474 sufficiently captured by *n*-gram based metrics. The RIBES metric applies rank corre-
9475 lation to measure the similarity in word order between the system and reference transla-
9476 tions (Isozaki et al., 2010).

9477 18.1.2 Data

9478 Data-driven approaches to machine translation rely primarily on **parallel corpora**: sentence-
9479 level translations. Early work focused on government records, in which fine-grained offi-
9480 cial translations are often required. For example, the IBM translation systems were based
9481 on the proceedings of the Canadian Parliament, called **Hansards**, which are recorded in
9482 English and French (Brown et al., 1990). The growth of the European Union led to the
9483 development of the **EuroParl corpus**, which spans 21 European languages (Koehn, 2005).
9484 While these datasets helped to launch the field of machine translation, they are restricted
9485 to narrow domains and a formal speaking style, limiting their applicability to other types
9486 of text. As more resources are committed to machine translation, new translation datasets
9487 have been commissioned. This has broadened the scope of available data to news,² movie
9488 subtitles,³ social media (Ling et al., 2013), dialogues (Fordyce, 2007), TED talks (Paul et al.,
9489 2010), and scientific research articles (Nakazawa et al., 2016).

9490 Despite this growing set of resources, the main bottleneck in machine translation data
9491 is the need for parallel corpora that are aligned at the sentence level. Many languages have
9492 sizable parallel corpora with some high-resource language, but not with each other. The
9493 high-resource language can then be used as a “pivot” or “bridge” (Boitet, 1988; Utiyama

²https://catalog.ldc.upenn.edu/LDC2010T10_translation-task.html <http://www.statmt.org/wmt15/>

³<http://opus.nlpl.eu/>

and Isahara, 2007): for example, De Gispert and Marino (2006) use Spanish as a bridge for translation between Catalan and English. For most of the 6000 languages spoken today, the only source of translation data remains the Judeo-Christian Bible (Resnik et al., 1999). While relatively small, at less than a million tokens, the Bible has been translated into more than 2000 languages, far outpacing any other corpus. Some research has explored the possibility of automatically identifying parallel sentence pairs from unaligned parallel texts, such as web pages and Wikipedia articles (Kilgarriff and Grefenstette, 2003; Resnik and Smith, 2003; Adafre and De Rijke, 2006). Another approach is to create large parallel corpora through crowdsourcing (Zaidan and Callison-Burch, 2011).

18.2 Statistical machine translation

The previous section introduced adequacy and fluency as the two main criteria for machine translation. A natural modeling approach is to represent them with separate scores,

$$\Psi(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \Psi_A(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) + \Psi_F(\mathbf{w}^{(t)}). \quad [18.3]$$

The fluency score Ψ_F need not even consider the source sentence; it only judges $\mathbf{w}^{(t)}$ on whether it is fluent in the target language. This decomposition is advantageous because it makes it possible to estimate the two scoring functions on separate data. While the adequacy model must be estimated from aligned sentences — which are relatively expensive and rare — the fluency model can be estimated from monolingual text in the target language. Large monolingual corpora are now available in many languages, thanks to resources such as Wikipedia.

An elegant justification of the decomposition in Equation 18.3 is provided by the **noisy channel model**, in which each scoring function is a log probability:

$$\Psi_A(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) \triangleq \log p_{S|T}(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}) \quad [18.4]$$

$$\Psi_F(\mathbf{w}^{(t)}) \triangleq \log p_T(\mathbf{w}^{(t)}) \quad [18.5]$$

$$\Psi(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \log p_{S|T}(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}) + \log p_T(\mathbf{w}^{(t)}) = \log p_{S,T}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}). \quad [18.6]$$

By setting the scoring functions equal to the logarithms of the prior and likelihood, their sum is equal to $\log p_{S,T}$, which is the logarithm of the joint probability of the source and target. The sentence $\hat{\mathbf{w}}^{(t)}$ that maximizes this joint probability is also the maximizer of the conditional probability $p_{T|S}$, making it the most likely target language sentence, conditioned on the source.

The noisy channel model can be justified by a generative story. The target text is originally generated from a probability model p_T . It is then encoded in a “noisy channel” $p_{S|T}$, which converts it to a string in the source language. In decoding, we apply Bayes’ rule to recover the string $\mathbf{w}^{(t)}$ that is maximally likely under the conditional probability

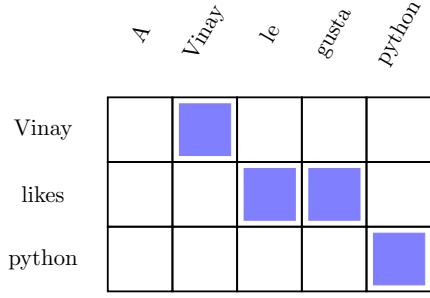


Figure 18.3: An example word-to-word alignment

9522 $p_{T|S}$. Under this interpretation, the target probability p_T is just a language model, and
 9523 can be estimated using any of the techniques from chapter 6. The only remaining learning
 9524 problem is to estimate the translation model $p_{S|T}$.

9525 18.2.1 Statistical translation modeling

9526 The simplest decomposition of the translation model is word-to-word: each word in the
 9527 source should be aligned to a word in the translation. This approach presupposes an
 9528 **alignment** $\mathcal{A}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)})$, which contains a list of pairs of source and target tokens. For
 9529 example, given $\mathbf{w}^{(s)} = A\ Vinay\ le\ gusta\ Python$ and $\mathbf{w}^{(t)} = Vinay\ likes\ Python$, one possible
 9530 word-to-word alignment is,

$$\mathcal{A}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \{(A, \emptyset), (Vinay, Vinay), (le, likes), (gusta, likes), (Python, Python)\}. \quad [18.7]$$

9531 This alignment is shown in Figure 18.3. Another, less promising, alignment is:

$$\mathcal{A}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \{(A, Vinay), (Vinay, likes), (le, Python), (gusta, \emptyset), (Python, \emptyset)\}. \quad [18.8]$$

9532 Each alignment contains exactly one tuple for each word in the *source*, which serves to
 9533 explain how the source word could be translated from the target, as required by the trans-
 9534 lation probability $p_{S|T}$. If no appropriate word in the target can be identified for a source
 9535 word, it is aligned to \emptyset — as is the case for the Spanish function word *a* in the example,
 9536 which glosses to the English word *to*. Words in the target can align with multiple words
 9537 in the source, so that the target word *likes* can align to both *le* and *gusta* in the source.

The joint probability of the alignment and the translation can be defined conveniently

as,

$$p(\mathbf{w}^{(s)}, \mathcal{A} | \mathbf{w}^{(t)}) = \prod_{m=1}^{M^{(s)}} p(w_m^{(s)}, a_m | w_{a_m}^{(t)}, m, M^{(s)}, M^{(t)}) \quad [18.9]$$

$$= \prod_{m=1}^{M^{(s)}} p(a_m | m, M^{(s)}, M^{(t)}) \times p(w_m^{(s)} | w_{a_m}^{(t)}). \quad [18.10]$$

9538 This probability model makes two key assumptions:

9539 • The alignment probability factors across tokens,

$$p(\mathcal{A} | \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \prod_{m=1}^{M^{(s)}} p(a_m | m, M^{(s)}, M^{(t)}). \quad [18.11]$$

9540 This means that each alignment decision is independent of the others, and depends
9541 only on the index m , and the sentence lengths $M^{(s)}$ and $M^{(t)}$.

9542 • The translation probability also factors across tokens,

$$p(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}, \mathcal{A}) = \prod_{m=1}^{M^{(s)}} p(w_m^{(s)} | w_{a_m}^{(t)}), \quad [18.12]$$

9543 so that each word in $\mathbf{w}^{(s)}$ depends only on its aligned word in $\mathbf{w}^{(t)}$. This means that
9544 translation is word-to-word, ignoring context. The hope is that the target language
9545 model $p(\mathbf{w}^{(t)})$ will correct any disfluencies that arise from word-to-word translation.

To translate with such a model, we could sum or max over all possible alignments,

$$p(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \sum_{\mathcal{A}} p(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}, \mathcal{A}) \quad [18.13]$$

$$= p(\mathbf{w}^{(t)}) \sum_{\mathcal{A}} p(\mathcal{A}) \times p(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}, \mathcal{A}) \quad [18.14]$$

$$\geq p(\mathbf{w}^{(t)}) \max_{\mathcal{A}} p(\mathcal{A}) \times p(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}, \mathcal{A}). \quad [18.15]$$

The term $p(\mathcal{A})$ defines the prior probability over alignments. A series of alignment models with increasingly relaxed independence assumptions was developed by researchers at IBM in the 1980s and 1990s, known as IBM Models 1-6 (Och and Ney, 2003). IBM Model 1 makes the strongest independence assumption:

$$p(a_m | m, M^{(s)}, M^{(t)}) = \frac{1}{M^{(t)}}. \quad [18.16]$$

9546 In this model, every alignment is equally likely. This is almost surely wrong, but it re-
 9547 sults in a convex learning objective, yielding a good initialization for the more complex
 9548 alignment models (Brown et al., 1993; Koehn, 2009).

9549 18.2.2 Estimation

9550 Let us define the parameter $\theta_{u \rightarrow v}$ as the probability of translating target word u to source
 9551 word v . If word-to-word alignments were annotated, these probabilities could be com-
 9552 puted from relative frequencies,

$$\hat{\theta}_{u \rightarrow v} = \frac{\text{count}(u, v)}{\text{count}(u)}, \quad [18.17]$$

9553 where $\text{count}(u, v)$ is the count of instances in which word v was aligned to word u in
 9554 the training set, and $\text{count}(u)$ is the total count of the target word u . The smoothing
 9555 techniques mentioned in chapter 6 can help to reduce the variance of these probability
 9556 estimates.

9557 Conversely, if we had an accurate translation model, we could estimate the likelihood
 9558 of each alignment decision,

$$q_m(a_m | \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) \propto p(a_m | m, M^{(s)}, M^{(t)}) \times p(w_m^{(s)} | w_{a_m}^{(t)}), \quad [18.18]$$

where $q_m(a_m | \mathbf{w}^{(s)}, \mathbf{w}^{(t)})$ is a measure of our confidence in aligning source word $w_m^{(s)}$
 to target word $w_{a_m}^{(t)}$. The relative frequencies could then be computed from the *expected
 counts*,

$$\hat{\theta}_{u \rightarrow v} = \frac{E_q[\text{count}(u, v)]}{\text{count}(u)} \quad [18.19]$$

$$E_q[\text{count}(u, v)] = \sum_m q_m(a_m | \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) \delta(w_m^{(s)} = v) \delta(w_{a_m}^{(t)} = u). \quad [18.20]$$

9559 The **expectation-maximization** (EM) algorithm proceeds by iteratively updating q_m
 9560 and $\hat{\Theta}$. The algorithm is described in general form in chapter 5. For statistical machine
 9561 translation, the steps of the algorithm are:

- 9562 1. **E-step:** Update beliefs about word alignment using Equation 18.18.
- 9563 2. **M-step:** Update the translation model using Equations 18.19 and 18.20.

9564 As discussed in chapter 5, the expectation maximization algorithm is guaranteed to con-
 9565 verge, but not to a global optimum. However, for IBM Model 1, it can be shown that EM
 9566 optimizes a convex objective, and global optimality is guaranteed. For this reason, IBM
 9567 Model 1 is often used as an initialization for more complex alignment models. For more
 9568 detail, see Koehn (2009).

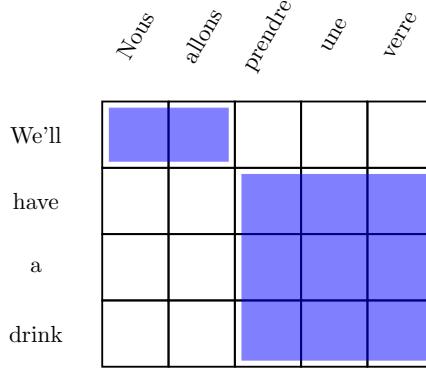


Figure 18.4: A phrase-based alignment between French and English, corresponding to example (18.3)

9569 18.2.3 Phrase-based translation

9570 Real translations are not word-to-word substitutions. One reason is that many multiword
 9571 expressions are not translated literally, as shown in this example from French:

- 9572 (18.3) *Nous allons prendre un verre*
 We will take a glass
 9573 We'll have a drink

9574 The line *we will take a glass* is the word-for-word gloss of the French sentence; the transla-
 9575 tion *we'll have a drink* is shown on the third line. Such examples are difficult for word-to-
 9576 word translation models, since they require translating *prendre* to *have* and *verre* to *drink*.
 9577 These translations are only correct in the context of these specific phrases.

Phrase-based translation generalizes on word-based models by building translation tables and alignments between multiword spans. (These “phrases” are not necessarily syntactic constituents like the noun phrases and verb phrases described in chapters 9 and 10.) The generalization from word-based translation is surprisingly straightforward: the translation tables can now condition on multi-word units, and can assign probabilities to multi-word units; alignments are mappings from spans to spans, $((i, j), (k, \ell))$, so that

$$p(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}, \mathcal{A}) = \prod_{((i, j), (k, \ell)) \in \mathcal{A}} p_{w^{(s)}|w^{(t)}}(\{w_{i+1}^{(s)}, w_{i+2}^{(s)}, \dots, w_j^{(s)}\} | \{w_{k+1}^{(t)}, w_{k+2}^{(t)}, \dots, w_\ell^{(t)}\}). \quad [18.21]$$

9578 The phrase alignment $((i, j), (k, \ell))$ indicates that the span $\mathbf{w}_{i+1:j}^{(s)}$ is the translation of the
 9579 span $\mathbf{w}_{k+1:\ell}^{(t)}$. An example phrasal alignment is shown in Figure 18.4. Note that the align-

ment set \mathcal{A} is required to cover all of the tokens in the source, just as in word-based translation. The probability model $p_{w^{(s)}|w^{(t)}}$ must now include translations for all phrase pairs, which can be learned from expectation-maximization just as in word-based statistical machine translation.

18.2.4 *Syntax-based translation

The Vauquois Pyramid (Figure 18.1) suggests that translation might be easier if we take a higher-level view. One possibility is to incorporate the syntactic structure of the source, the target, or both. This is particularly promising for language pairs that consistent syntactic differences. For example, English adjectives almost always precede the nouns that they modify, while in Romance languages such as French and Spanish, the adjective often follows the noun: thus, *angry fish* would translate to *pez (fish) enojado (angry)* in Spanish. In word-to-word translation, these reorderings cause the alignment model to be overly permissive. It is not that the order of *any* pair of English words can be reversed when translating into Spanish, but only adjectives and nouns within a noun phrase. Similar issues arise when translating between verb-final languages such as Japanese (in which verbs usually follow the subject and object), verb-initial languages like Tagalog and classical Arabic, and verb-medial languages such as English.

An elegant solution is to link parsing and translation in a **synchronous context-free grammar** (SCFG; Chiang, 2007).⁴ An SCFG is a set of productions of the form $X \rightarrow (\alpha, \beta, \sim)$, where X is a non-terminal, α and β are sequences of terminals or non-terminals, and \sim is a one-to-one alignment of items in α with items in β . To handle the English-Spanish adjective-noun ordering, an SCFG would include productions such as,

$$\text{NP} \rightarrow (\text{DET}_1 \text{NN}_2 \text{JJ}_3, \quad \text{DET}_1 \text{JJ}_3 \text{NN}_2), \quad [18.22]$$

with subscripts indicating the alignment between the Spanish (left) and English (right) parts of the right-hand side. Terminal productions yield translation pairs,

$$\text{JJ} \rightarrow (\text{enojado}_1, \text{angry}_1). \quad [18.23]$$

A synchronous derivation begins with the start symbol S , and derives a pair of sequences of terminal symbols.

Given an SCFG in which each production yields at most two symbols in each language (Chomsky Normal Form; see § 9.2.1.2), a sentence can be parsed using only the CKY algorithm (chapter 10). The resulting derivation also includes productions in the other language, all the way down to the surface form. Therefore, SCFGs make translation very similar to parsing. In a weighted SCFG, the log probability $\log p_{S|T}$ can be computed from

⁴Key earlier work includes syntax-driven transduction (Lewis II and Stearns, 1968) and stochastic inversion transduction grammars (Wu, 1997).

the sum of the log-probabilities of the productions. However, combining SCFGs with a target language model is computationally expensive, necessitating approximate search algorithms (Huang and Chiang, 2007).

Synchronous context-free grammars are an example of **tree-to-tree translation**, because they model the syntactic structure of both the target and source language. In **string-to-tree translation**, string elements are translated into constituent tree fragments, which are then assembled into a translation (Yamada and Knight, 2001; Galley et al., 2004); in **tree-to-string translation**, the source side is parsed, and then transformed into a string on the target side (Liu et al., 2006). A key question for syntax-based translation is the extent to which we phrasal constituents align across translations (Fox, 2002), because this governs the extent to which we can rely on monolingual parsers and treebanks. For more on syntax-based machine translation, see the monograph by Williams et al. (2016).

18.3 Neural machine translation

Neural network models for machine translation are based on the **encoder-decoder** architecture (Cho et al., 2014). The encoder network converts the source language sentence into a vector or matrix representation; the decoder network then converts the encoding into a sentence in the target language.

$$z = \text{ENCODE}(\mathbf{w}^{(s)}) \quad [18.24]$$

$$\mathbf{w}^{(t)} \mid \mathbf{w}^{(s)} \sim \text{DECODE}(z), \quad [18.25]$$

where the second line means that the function $\text{DECODE}(z)$ defines the conditional probability $p(\mathbf{w}^{(t)} \mid \mathbf{w}^{(s)})$.

The decoder is typically a recurrent neural network, which generates the target language sentence one word at a time, while recurrently updating a hidden state. The encoder and decoder networks are trained end-to-end from parallel sentences. If the output layer of the decoder is a logistic function, then the entire architecture can be trained to maximize the conditional log-likelihood,

$$\log p(\mathbf{w}^{(t)} \mid \mathbf{w}^{(s)}) = \sum_{m=1}^{M^{(t)}} p(w_m^{(t)} \mid \mathbf{w}_{1:m-1}^{(t)}, z) \quad [18.26]$$

$$p(w_m^{(t)} \mid \mathbf{w}_{1:m-1}^{(t)}, \mathbf{w}^{(s)}) \propto \exp \left(\boldsymbol{\beta}_{w_m^{(t)}} \cdot \mathbf{h}_{m-1}^{(t)} \right) \quad [18.27]$$

where the hidden state $\mathbf{h}_{m-1}^{(t)}$ is a recurrent function of the previously generated text $\mathbf{w}_{1:m-1}^{(t)}$ and the encoding z . The second line is equivalent to writing,

$$w_m^{(t)} \mid \mathbf{w}_{1:m-1}^{(t)}, \mathbf{w}^{(s)} \sim \text{SoftMax} \left(\boldsymbol{\beta} \cdot \mathbf{h}_{m-1}^{(t)} \right), \quad [18.28]$$

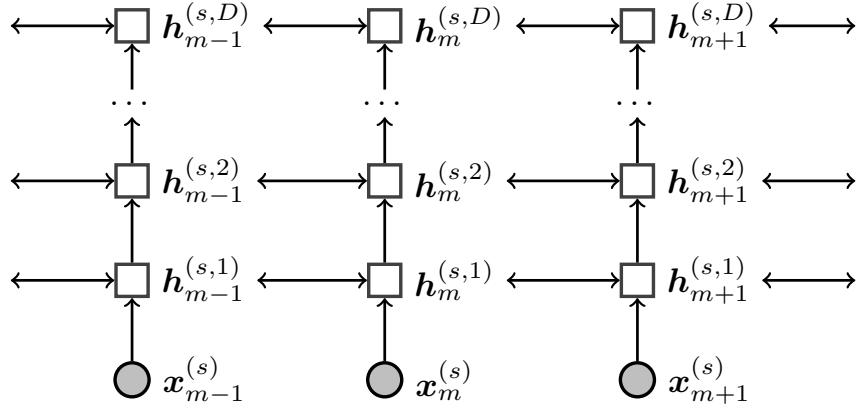


Figure 18.5: A deep bidirectional LSTM encoder

9626 where $\beta \in \mathbb{R}^{(V^{(t)} \times K)}$ is the matrix of output word vectors for the $V^{(t)}$ words in the target
 9627 language vocabulary.

The simplest encoder-decoder architecture is the **sequence-to-sequence** model (Sutskever et al., 2014). In this model, the encoder is set to the final hidden state of a **long short-term memory (LSTM)** (see § 6.3.3) on the source sentence:

$$\mathbf{h}_m^{(s)} = \text{LSTM}(\mathbf{x}_m^{(s)}, \mathbf{h}_{m-1}^{(s)}) \quad [18.29]$$

$$\mathbf{z} \triangleq \mathbf{h}_{M^{(s)}}^{(s)}, \quad [18.30]$$

where $\mathbf{x}_m^{(s)}$ is the embedding of source language word $w_m^{(s)}$. The encoding then provides the initial hidden state for the decoder LSTM:

$$\mathbf{h}_0^{(t)} = \mathbf{z} \quad [18.31]$$

$$\mathbf{h}_m^{(t)} = \text{LSTM}(\mathbf{x}_m^{(t)}, \mathbf{h}_{m-1}^{(t)}), \quad [18.32]$$

9628 where $\mathbf{x}_m^{(t)}$ is the embedding of the target language word $w_m^{(t)}$.

9629 Sequence-to-sequence translation is nothing more than wiring together two LSTMs:
 9630 one to read the source, and another to generate the target. To make the model work well,
 9631 some additional tweaks are needed:

- 9632 • Most notably, the model works much better if the source sentence is reversed, reading
 9633 from the end of the sentence back to the beginning. In this way, the words at the
 9634 beginning of the source have the greatest impact on the encoding \mathbf{z} , and therefore

9635 impact the words at the beginning of the target sentence. Later work on more ad-
 9636 vanced encoding models, such as **neural attention** (see § 18.3.1), has eliminated the
 9637 need for reversing the source sentence.

- The encoder and decoder can be implemented as **deep LSTMs**, with multiple layers of hidden states. As shown in Figure 18.5, each hidden state $\mathbf{h}_m^{(s,i)}$ at layer i is treated as the input to an LSTM at layer $i + 1$:

$$\mathbf{h}_m^{(s,1)} = \text{LSTM}(\mathbf{x}_m^{(s)}, \mathbf{h}_{m-1}^{(s)}) \quad [18.33]$$

$$\mathbf{h}_m^{(s,i+1)} = \text{LSTM}(\mathbf{h}_m^{(s,i)}, \mathbf{h}_{m-1}^{(s)}), \quad \forall i \geq 1. \quad [18.34]$$

9638 The original work on sequence-to-sequence translation used four layers; in 2016,
 9639 Google’s commercial machine translation system used eight layers (Wu et al., 2016).⁵

- 9640 • Significant improvements can be obtained by creating an **ensemble** of translation
 9641 models, each trained from a different random initialization. For an ensemble of size
 9642 N , the per-token decoding probability is set equal to,

$$p(w^{(t)} | \mathbf{z}, \mathbf{w}_{1:m-1}^{(t)}) = \frac{1}{N} \sum_{i=1}^N p_i(w^{(t)} | \mathbf{z}, \mathbf{w}_{1:m-1}^{(t)}), \quad [18.35]$$

9643 where p_i is the decoding probability for model i . Each translation model in the
 9644 ensemble includes its own encoder and decoder networks.

- 9645 • The original sequence-to-sequence model used a fairly standard training setup: stochas-
 9646 tic gradient descent with an exponentially decreasing learning rate after the first five
 9647 epochs; mini-batches of 128 sentences, chosen to have similar length so that each
 9648 sentence on the batch will take roughly the same amount of time to process; gradi-
 9649 ent clipping (see § 3.3.4) to ensure that the norm of the gradient never exceeds some
 9650 predefined value.

9651 18.3.1 Neural attention

9652 The sequence-to-sequence model discussed in the previous section was a radical depart-
 9653 ure from statistical machine translation, in which each word or phrase in the target lan-
 9654 guage is conditioned on a single word or phrase in the source language. Both approaches
 9655 have advantages. Statistical translation leverages the idea of compositionality — transla-
 9656 tions of large units should be based on the translations of their component parts — and
 9657 this seems crucial if we are to scale translation to longer units of text. But the translation
 9658 of each word or phrase often depends on the larger context, and encoder-decoder models
 9659 capture this context at the sentence level.

⁵Google reports that this system took six days to train for English-French translation, using 96 NVIDIA K80 GPUs, which would have cost roughly half a million dollars at the time.

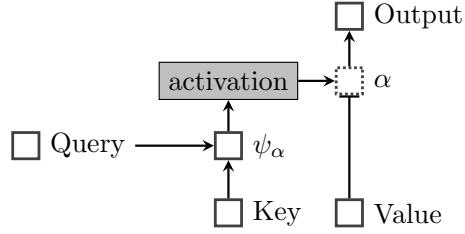


Figure 18.6: A general view of neural attention. The dotted box indicates that each $\alpha_{m \rightarrow n}$ can be viewed as a **gate** on value n .

Is it possible for translation to be both contextualized and compositional? One approach is to augment neural translation with an **attention mechanism**. The idea of neural attention was described in § 17.5, but its application to translation bears further discussion. In general, attention can be thought of as using a query to select from a memory of key-value pairs. However, the query, keys, and values are all vectors, and the entire operation is differentiable. For each key n in the memory, we compute a score $\psi_\alpha(m, n)$ with respect to the query m . That score is a function of the compatibility of the key and the query, and can be computed using a small feedforward neural network. The vector of scores is passed through an activation function, such as softmax. The output of this activation function is a vector of non-negative numbers $[\alpha_{m \rightarrow 1}, \alpha_{m \rightarrow 2}, \dots, \alpha_{m \rightarrow N}]^\top$, with length N equal to the size of the memory. Each value in the memory v_n is multiplied by the attention $\alpha_{m \rightarrow n}$; the sum of these scaled values is the output. This process is shown in Figure 18.6. In the extreme case that $\alpha_{m \rightarrow n} = 1$ and $\alpha_{m \rightarrow n'} = 0$ for all other n' , then the attention mechanism simply selects the value v_n from the memory.

Neural attention makes it possible to integrate alignment into the encoder-decoder architecture. Rather than encoding the entire source sentence into a fixed length vector z , it can be encoded into a matrix $Z \in \mathbb{R}^{K \times M^{(S)}}$, where K is the dimension of the hidden state, and $M^{(S)}$ is the number of tokens in the source input. Each column of Z represents the state of a recurrent neural network over the source sentence. These vectors are constructed from a **bidirectional LSTM** (see § 7.6), which can be a deep network as shown in Figure 18.5. These columns are both the keys and the values in the attention mechanism.

At each step m in decoding, the attentional state is computed by executing a query, which is equal to the state of the decoder, $h_m^{(t)}$. The resulting compatibility scores are,

$$\psi_\alpha(m, n) = v_\alpha \cdot \tanh(\Theta_\alpha[h_m^{(t)}; h_n^{(s)}]). \quad [18.36]$$

The function ψ is thus a two layer feedforward neural network, with weights v_α on the output layer, and weights Θ_α on the input layer. To convert these scores into attention weights, we apply an activation function, which can be vector-wise softmax or an

9684 element-wise sigmoid:

Softmax attention

$$\alpha_{m \rightarrow n} = \frac{\exp \psi_\alpha(m, n)}{\sum_{n'=1}^{M^{(s)}} \exp \psi_\alpha(m, n')} \quad [18.37]$$

Sigmoid attention

$$\alpha_{m \rightarrow n} = \sigma(\psi_\alpha(m, n)) \quad [18.38]$$

The attention α is then used to compute an **context vector** c_m by taking a weighted average over the columns of Z ,

$$c_m = \sum_{n=1}^{M^{(s)}} \alpha_{m \rightarrow n} z_n, \quad [18.39]$$

where $\alpha_{m \rightarrow n} \in [0, 1]$ is the amount of attention from word m of the target to word n of the source. The context vector can be incorporated into the decoder's word output probability model, by adding another layer to the decoder (Luong et al., 2015):

$$\tilde{h}_m^{(t)} = \tanh(\Theta_c[h_m^{(t)}; c_m]) \quad [18.40]$$

$$p(w_{m+1}^{(t)} | w_{1:m}^{(t)}, w^{(s)}) \propto \exp\left(\beta_{w_{m+1}^{(t)}} \cdot \tilde{h}_m^{(t)}\right). \quad [18.41]$$

9685 Here the decoder state $h_m^{(t)}$ is concatenated with the context vector, forming the input
 9686 to compute a final output vector $\tilde{h}_m^{(t)}$. The context vector can be incorporated into the
 9687 decoder recurrence in a similar manner (Bahdanau et al., 2014).

9688 **18.3.2 *Neural machine translation without recurrence**

In the encoder-decoder model, attention's “keys and values” are the hidden state representations in the encoder network, z , and the “queries” are state representations in the decoder network $h^{(t)}$. It is also possible to completely eliminate recurrence from neural translation, by applying **self-attention** (Lin et al., 2017; Kim et al., 2017) within the encoder and decoder, as in the **transformer architecture** (Vaswani et al., 2017). For level i , the basic equations of the encoder side of the transformer are:

$$z_m^{(i)} = \sum_{n=1}^{M^{(s)}} \alpha_{m \rightarrow n}^{(i)} (\Theta_v h_n^{(i-1)}) \quad [18.42]$$

$$h_m^{(i)} = \Theta_2 \text{ReLU}\left(\Theta_1 z_m^{(i)} + b_1\right) + b_2. \quad [18.43]$$

9689 For each token m at level i , we compute self-attention over the entire source sentence:
 9690 the keys, values, and queries are all projections of the vector $\mathbf{h}^{(i-1)}$. The attention scores
 9691 $\alpha_{m \rightarrow n}^{(i)}$ are computed using a scaled form of softmax attention,

$$\alpha_{m \rightarrow n} \propto \exp(\psi_\alpha(m, n)/M), \quad [18.44]$$

9692 where M is the length of the input. This encourages the attention to be more evenly
 9693 dispersed across the input. Self-attention is applied across multiple “heads”, each using
 9694 different projections of $\mathbf{h}^{(i-1)}$ to form the keys, values, and queries.

9695 The output of the self-attentional layer is the representation $\mathbf{z}_m^{(i)}$, which is then passed
 9696 through a two-layer feed-forward network, yielding the input to the next layer, $\mathbf{h}^{(i)}$. To
 9697 ensure that information about word order in the source is integrated into the model, the
 9698 encoder includes **positional encodings** of the index of each word in the source. These
 9699 encodings are vectors for each position $m \in \{1, 2, \dots, M\}$. The positional encodings are
 9700 concatenated with the word embeddings \mathbf{x}_m at the base layer of the model.⁶

9701 Convolutional neural networks (see § 3.4) have also been applied as encoders in neu-
 9702 ral machine translation. For each word $w_m^{(s)}$, a convolutional network computes a rep-
 9703 resentation $\mathbf{h}_m^{(s)}$ from the embeddings of the word and its neighbors. This procedure is
 9704 applied several times, creating a deep convolutional network. The recurrent decoder then
 9705 computes a set of attention weights over these convolutional representations, using the
 9706 decoder’s hidden state $\mathbf{h}^{(t)}$ as the queries. This attention vector is used to compute a
 9707 weighted average over the outputs of *another* convolutional neural network of the source,
 9708 yielding an averaged representation c_m , which is then fed into the decoder. As with the
 9709 transformer, speed is the main advantage over recurrent encoding models; another sim-
 9710 ilarity is that word order information is approximated through the use of positional en-
 9711 codings. It seems likely that there are limitations to how well positional encodings can
 9712 account for word order and deeper linguistic structure. But for the moment, the com-
 9713 putational advantages of such approaches have put them on par with the best recurrent
 9714 translation models.⁷

9715 18.3.3 Out-of-vocabulary words

9716 Thus far, we have treated translation as a problem at the level of words or phrases. For
 9717 words that do not appear in the training data, all such models will struggle. There are
 9718 two main reasons for the presence of out-of-vocabulary (OOV) words:

⁶The transformer architecture relies on several additional tricks, including **layer normalization** (see § 3.3.4) and residual connections around the nonlinear activations (see § 3.2.2).

⁷A recent evaluation found that best performance was obtained by using a recurrent network for the decoder, and a transformer for the encoder (Chen et al., 2018). The transformer was also found to significantly outperform a convolutional neural network.

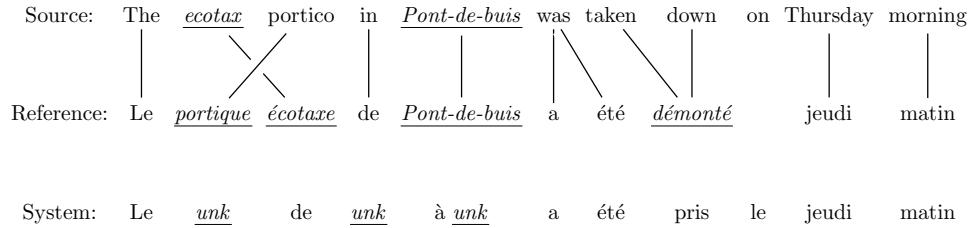


Figure 18.7: Translation with *unknown words*. The system outputs *unk* to indicate words that are outside its vocabulary. Figure adapted from Luong et al. (2015).

9719 • New proper nouns, such as family names or organizations, are constantly arising —
 9720 particularly in the news domain. The same is true, to a lesser extent, for technical
 9721 terminology. This issue is shown in Figure 18.7.

9722 • In many languages, words have complex internal structure, known as **morphology**.
 9723 An example is German, which uses compounding to form nouns like *Abwasserbe-*
 9724 *handlungsanlage* (*sewage water treatment plant*; example from Sennrich et al. (2016)).
 9725 While compounds could in principle be addressed by better tokenization (see § 8.4),
 9726 other morphological processes involve more complex transformations of subword
 9727 units.

9728 Names and technical terms can be handled in a postprocessing step: after first identi-
 9729 fying alignments between unknown words in the source and target, we can look up each
 9730 aligned source word in a dictionary, and choose a replacement (Luong et al., 2015). If the
 9731 word does not appear in the dictionary, it is likely to be a proper noun, and can be copied
 9732 directly from the source to the target. This approach can also be integrated directly into
 9733 the translation model, rather than applying it as a postprocessing step (Jean et al., 2015).

9734 Words with complex internal structure can be handled by translating subword units
 9735 rather than entire words. A popular technique for identifying subword units is **byte-pair**
 9736 **encoding** (BPE; Gage, 1994; Sennrich et al., 2016). The initial vocabulary is defined as the
 9737 set of characters used in the text. The most common character bigram is then merged into
 9738 a new symbol, and the vocabulary is updated. The merging operation is applied repeat-
 9739 edly, until the vocabulary reaches some maximum size. For example, given the dictionary
 9740 {fish, fished, want, wanted, bike, biked}, we would first merge e+d into the subword unit ed,
 9741 since this bigram appears in three words of the six words. Next, there are several bigrams
 9742 that each appear in a pair of words: f+i, i+s, s+h, w+a, a+n, etc. These can be merged
 9743 in any order, resulting in the segmentation, {fish, fish+ed, want, want+ed, bik+e, bik+ed}. At
 9744 this point, there are no subword bigrams that appear more than once. In real data, merg-
 9745 ing is performed until the number of subword units reaches some predefined threshold,

9746 such as 10^4 .

9747 Each subword unit is treated as a token for translation, in both the encoder (source
 9748 side) and decoder (target side). BPE can be applied jointly to the union of the source and
 9749 target vocabularies, identifying subword units that appear in both languages. For lan-
 9750 guages that have different scripts, such as English and Russian, **transliteration** between
 9751 the scripts should be applied first.⁸

9752 18.4 Decoding

Given a trained translation model, the decoding task is:

$$\hat{\mathbf{w}}^{(t)} = \underset{\mathbf{w} \in \mathcal{V}^*}{\operatorname{argmax}} \Psi(\mathbf{w}, \mathbf{w}^{(s)}), \quad [18.45]$$

9753 where $\mathbf{w}^{(t)}$ is a sequence of tokens from the target vocabulary \mathcal{V} . It is not possible to
 9754 efficiently obtain exact solutions to the decoding problem, for even minimally effective
 9755 models in either statistical or neural machine translation. Today's state-of-the-art transla-
 9756 tion systems use **beam search** (see § 11.3.1.4), which is an incremental decoding algorithm
 9757 that maintains a small constant number of competitive hypotheses. Such greedy approxi-
 9758 mations are reasonably effective in practice, and this may be in part because the decoding
 9759 objective is only loosely correlated with measures of translation quality, so that exact op-
 9760 timization of [18.45] may not greatly improve the resulting translations.

Decoding in neural machine translation is somewhat simpler than in phrase-based
 statistical machine translation.⁹ The scoring function Ψ is defined,

$$\Psi(\mathbf{w}^{(t)}, \mathbf{w}^{(s)}) = \sum_{m=1}^{M^{(t)}} \psi(w_m^{(t)}; \mathbf{w}_{1:m-1}^{(t)}, \mathbf{z}) \quad [18.46]$$

$$\psi(w^{(t)}; \mathbf{w}_{1:m-1}^{(t)}, \mathbf{z}) = \beta_{w_m^{(t)}} \cdot \mathbf{h}_m^{(t)} - \log \sum_{w \in \mathcal{V}} \exp(\beta_w \cdot \mathbf{h}_m^{(t)}), \quad [18.47]$$

9761 where \mathbf{z} is the encoding of the source sentence $\mathbf{w}^{(s)}$, and $\mathbf{h}_m^{(t)}$ is a function of the encoding
 9762 \mathbf{z} and the decoding history $\mathbf{w}_{1:m-1}^{(t)}$. This formulation subsumes the attentional translation
 9763 model, where \mathbf{z} is a matrix encoding of the source.

Now consider the incremental decoding algorithm,

$$\hat{w}_m^{(t)} = \underset{w \in \mathcal{V}}{\operatorname{argmax}} \psi(w; \hat{\mathbf{w}}_{1:m-1}^{(t)}, \mathbf{z}), \quad m = 1, 2, \dots \quad [18.48]$$

⁸Transliteration is crucial for converting names and other foreign words between languages that do not share a single script, such as English and Japanese. It is typically approached using the finite-state methods discussed in chapter 9 (Knight and Graehl, 1998).

⁹For more on decoding in phrase-based statistical models, see Koehn (2009).

9764 This algorithm selects the best target language word at position m , assuming that it has
 9765 already generated the sequence $\hat{w}_{1:m-1}^{(t)}$. (Termination can be handled by augmenting
 9766 the vocabulary \mathcal{V} with a special end-of-sequence token, ■.) The incremental algorithm
 9767 is likely to produce a suboptimal solution to the optimization problem defined in Equa-
 9768 tion 18.45, because selecting the highest-scoring word at position m can set the decoder
 9769 on a “garden path,” in which there are no good choices at some later position $n > m$. We
 9770 might hope for some dynamic programming solution, as in sequence labeling (§ 7.3). But
 9771 the Viterbi algorithm and its relatives rely on a Markov decomposition of the objective
 9772 function into a sum of local scores: for example, scores can consider locally adjacent tags
 9773 (y_m, y_{m-1}), but not the entire tagging history $y_{1:m}$. This decomposition is not applicable
 9774 to recurrent neural networks, because the hidden state $h_m^{(t)}$ is impacted by the entire his-
 9775 tory $w_{1:m}^{(t)}$; this sensitivity to long-range context is precisely what makes recurrent neural
 9776 networks so effective.¹⁰ In fact, it can be shown that decoding from any recurrent neural
 9777 network is NP-complete (Siegelmann and Sontag, 1995; Chen et al., 2018).

9778 **Beam search** Beam search is a general technique for avoiding search errors when ex-
 9779 haustive search is impossible; it was first discussed in § 11.3.1.4. Beam search can be
 9780 seen as a variant of the incremental decoding algorithm sketched in Equation 18.48, but
 9781 at each step m , a set of K different hypotheses are kept on the beam. For each hypothesis
 9782 $k \in \{1, 2, \dots, K\}$, we compute both the current score $\sum_{m=1}^{M^{(t)}} \psi(w_{k,m}^{(t)}; w_{k,1:m-1}^{(t)}, z)$ as well as
 9783 the current hidden state $h_k^{(t)}$. At each step in the beam search, the K top-scoring children
 9784 of each hypothesis currently on the beam are “expanded”, and the beam is updated. For
 9785 a detailed description of beam search for RNN decoding, see Graves (2012).

9786 **Learning and search** Conventionally, the learning algorithm is trained to predict the
 9787 right token in the translation, conditioned on the translation history being correct. But
 9788 if decoding must be approximate, then we might do better by modifying the learning
 9789 algorithm to be robust to errors in the translation history. **Scheduled sampling** does this
 9790 by training on histories that sometimes come from the ground truth, and sometimes come
 9791 from the model’s own output (Bengio et al., 2015).¹¹ As training proceeds, the training
 9792 wheels come off: we increase the fraction of tokens that come from the model rather than
 9793 the ground truth. Another approach is to train on an objective that relates directly to beam
 9794 search performance (Wiseman et al., 2016). **Reinforcement learning** has also been applied
 9795 to decoding of RNN-based translation models, making it possible to directly optimize
 9796 translation metrics such as BLEU (Ranzato et al., 2016).

¹⁰Note that this problem does not impact RNN-based sequence labeling models (see § 7.6). This is because the tags produced by these models do not affect the recurrent state.

¹¹Scheduled sampling builds on earlier work on learning to search (Daumé III et al., 2009; Ross et al., 2011), which are also described in § 15.2.4.

9797 18.5 Training towards the evaluation metric

9798 In likelihood-based training, the objective is to maximize the probability of a parallel
 9799 corpus. However, translations are not evaluated in terms of likelihood: metrics like BLEU
 9800 consider only the correctness of a single output translation, and not the range of prob-
 9801 abilities that the model assigns. It might therefore be better to train translation models
 9802 to achieve the highest BLEU score possible — to the extent that we believe BLEU mea-
 9803 sures translation quality. Unfortunately, BLEU and related metrics are not friendly for
 9804 optimization: they are discontinuous, non-differentiable functions of the parameters of
 9805 the translation model.

Consider an error function $\Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)})$, which measures the discrepancy between the system translation $\hat{\mathbf{w}}^{(t)}$ and the reference translation $\mathbf{w}^{(t)}$; this function could be based on BLEU or any other metric on translation quality. One possible criterion would be to select the parameters θ that minimize the error of the system's preferred translation,

$$\hat{\mathbf{w}}^{(t)} = \operatorname{argmax}_{\mathbf{w}^{(t)}} \Psi(\mathbf{w}^{(t)}, \mathbf{w}^{(s)}; \theta) \quad [18.49]$$

$$\hat{\theta} = \operatorname{argmin}_{\theta} \Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(s)}) \quad [18.50]$$

9806 However, identifying the top-scoring translation $\hat{\mathbf{w}}^{(t)}$ is usually intractable, as described
 9807 in the previous section. In **minimum error-rate training (MERT)**, $\hat{\mathbf{w}}^{(t)}$ is selected from a
 9808 set of candidate translations $\mathcal{Y}(\mathbf{w}^{(s)})$; this is typically a strict subset of all possible transla-
 9809 tions, so that it is only possible to optimize an approximation to the true error rate (Och
 9810 and Ney, 2003).

A further issue is that the objective function in Equation 18.50 is discontinuous and non-differentiable, due to the argmax over translations: an infinitesimal change in the parameters θ could cause another translation to be selected, with a completely different error. To address this issue, we can instead minimize the **risk**, which is defined as the expected error rate,

$$R(\theta) = E_{\hat{\mathbf{w}}^{(t)} | \mathbf{w}^{(s)}, \theta} [\Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)})] \quad [18.51]$$

$$= \sum_{\hat{\mathbf{w}}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})} p(\hat{\mathbf{w}}^{(t)} | \mathbf{w}^{(s)}) \times \Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)}). \quad [18.52]$$

9811 **Minimum risk training** minimizes the sum of $R(\theta)$ across all instances in the training set.

The risk can be generalized by exponentiating the translation probabilities,

$$\tilde{p}(\mathbf{w}^{(t)}; \theta, \alpha) \propto \left(p(\mathbf{w}^{(t)} | \mathbf{w}^{(s)}; \theta) \right)^\alpha \quad [18.53]$$

$$\tilde{R}(\theta) = \sum_{\hat{\mathbf{w}}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})} \tilde{p}(\hat{\mathbf{w}}^{(t)} | \mathbf{w}^{(s)}; \alpha, \theta) \times \Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)}) \quad [18.54]$$

where $\mathcal{Y}(\mathbf{w}^{(s)})$ is now the set of *all* possible translations for $\mathbf{w}^{(s)}$. Exponentiating the probabilities in this way is known as **annealing** (Smith and Eisner, 2006). When $\alpha = 1$, then $\tilde{R}(\boldsymbol{\theta}) = R(\boldsymbol{\theta})$; when $\alpha = \infty$, then $\tilde{R}(\boldsymbol{\theta})$ is equivalent to the sum of the errors of the maximum probability translations for each sentence in the dataset.

Clearly the set of candidate translations $\mathcal{Y}(\mathbf{w}^{(s)})$ is too large to explicitly sum over. Because the error function Δ generally does not decompose into smaller parts, there is no efficient dynamic programming solution to sum over this set. We can approximate the sum $\sum_{\mathbf{w}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})}$ with a sum over a finite number of samples, $\{\mathbf{w}_1^{(t)}, \mathbf{w}_2^{(t)}, \dots, \mathbf{w}_K^{(t)}\}$. If these samples were drawn uniformly at random, then the (annealed) risk would be approximated as (Shen et al., 2016),

$$\tilde{R}(\boldsymbol{\theta}) \approx \frac{1}{Z} \sum_{k=1}^K \tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)}; \boldsymbol{\theta}, \alpha) \times \Delta(\mathbf{w}_k^{(t)}, \mathbf{w}^{(t)}) \quad [18.55]$$

$$Z = \sum_{k=1}^K \tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)}; \boldsymbol{\theta}, \alpha). \quad [18.56]$$

Shen et al. (2016) report that performance plateaus at $K = 100$ for minimum risk training of neural machine translation.

Uniform sampling over the set of all possible translations is undesirable, because most translations have very low probability. A solution from Monte Carlo estimation is **importance sampling**, in which we draw samples from a **proposal distribution** $q(\mathbf{w}^{(t)})$. This distribution can be set equal to the current translation model $p(\mathbf{w}^{(t)} | \mathbf{w}^{(s)}; \boldsymbol{\theta})$. Each sample is then weighted by an **importance score**, $\omega_k = \frac{\tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)})}{q(\mathbf{w}_k^{(t)})}$. The effect of this weighting is to correct for any mismatch between the proposal distribution q and the true distribution \tilde{p} . The risk can then be approximated as,

$$\mathbf{w}_k^{(t)} \sim q(\mathbf{w}^{(t)}) \quad [18.57]$$

$$\omega_k = \frac{\tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)})}{q(\mathbf{w}_k^{(t)})} \quad [18.58]$$

$$\tilde{R}(\boldsymbol{\theta}) \approx \frac{1}{\sum_{k=1}^K \omega_k} \sum_{k=1}^K \omega_k \times \Delta(\mathbf{w}_k^{(t)}, \mathbf{w}^{(t)}). \quad [18.59]$$

Importance sampling will generally give a more accurate approximation with a given number of samples. The only formal requirement is that the proposal assigns non-zero probability to every $\mathbf{w}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})$. For more on importance sampling and related methods, see Robert and Casella (2013).

9822 Additional resources

9823 A complete textbook on machine translation is available from Koehn (2009). While this
9824 book precedes recent work on neural translation, a more recent draft chapter on neural
9825 translation models is also available (Koehn, 2017). Neubig (2017) provides a comprehen-
9826 sive tutorial on neural machine translation, starting from first principles. The course notes
9827 from Cho (2015) are also useful.

9828 Several neural machine translation systems are available, in connection with each of
9829 the major neural computing libraries: `lamtram` is an implementation of neural machine
9830 translation in the `dynet` (Neubig et al., 2017); `OpenNMT` (Klein et al., 2017) is an imple-
9831 mentation primarily in `Torch`; `tensor2tensor` is an implementation of several of the
9832 Google translation models in `tensorflow` (Abadi et al., 2016).

9833 Literary translation is especially challenging, even for expert human translators. Mes-
9834 sud (2014) describes some of these issues in her review of an English translation of *L'étranger*,
9835 the 1942 French novel by Albert Camus.¹² She compares the new translation by Sandra
9836 Smith against earlier translations by Stuart Gilbert and Matthew Ward, focusing on the
9837 difficulties presented by a single word in the first sentence:

9838 Then, too, Smith has reconsidered the book's famous opening. Camus's
9839 original is deceptively simple: "*Aujourd'hui, maman est morte.*" Gilbert influ-
9840 enced generations by offering us "Mother died today"—inscribing in Meur-
9841 sault [the narrator] from the outset a formality that could be construed as
9842 heartlessness. But *maman*, after all, is intimate and affectionate, a child's name
9843 for his mother. Matthew Ward concluded that it was essentially untranslatable
9844 ("mom" or "mummy" being not quite apt), and left it in the original French:
9845 "Maman died today." There is a clear logic in this choice; but as Smith has
9846 explained, in an interview in *The Guardian*, *maman* "didn't really tell the reader
9847 anything about the connotation." She, instead, has translated the sentence as
9848 "My mother died today."

9849 I chose "My mother" because I thought about how someone would
9850 tell another person that his mother had died. Meursault is speaking
9851 to the reader directly. "My mother died today" seemed to me the
9852 way it would work, and also implied the closeness of "maman" you
9853 get in the French.

9854 Elsewhere in the book, she has translated *maman* as "mama" — again, striving
9855 to come as close as possible to an actual, colloquial word that will carry the
9856 same connotations as *maman* does in French.

¹²The book review is currently available online at <http://www.nybooks.com/articles/2014/06/05/camus-new-letranger/>.

9857 The passage is a useful reminder that while the quality of machine translation has
9858 improved dramatically in recent years, expert human translations draw on considerations
9859 that are beyond the ken of any known computational approach.

9860 **Exercises**

9861 1. Give a synchronized derivation (§ 18.2.4) for the Spanish-English translation,

9862 (18.4) *El pez enojado atacado.*
 The fish angry attacked.
9863 The angry fish attacked.

9864 As above, the second line shows a word-for-word gloss, and the third line shows
9865 the desired translation. Use the synchronized production rule in [18.22], and design
9866 the other production rules necessary to derive this sentence pair. You may derive
9867 (*atacado*, *attacked*) directly from VP.

9868 Chapter 19

9869 Text generation

9870 In many of the most interesting problems in natural language processing, language is
9871 the output. The previous chapter described the specific case of machine translation, but
9872 there are many other applications, from summarization of research articles, to automated
9873 journalism, to dialogue systems. This chapter emphasizes three main scenarios: data-to-
9874 text, in which text is generated to explain or describe a structured record or unstructured
9875 perceptual input; text-to-text, which typically involves fusing information from multiple
9876 linguistic sources into a single coherent summary; and dialogue, in which text is generated
9877 as part of an interactive conversation with one or more human participants.

9878 19.1 Data-to-text generation

9879 In data-to-text generation, the input ranges from structured records, such as the descrip-
9880 tion of an weather forecast (as shown in Figure 19.1), to unstructured perceptual data,
9881 such as a raw image or video; the output may be a single sentence, such as an image cap-
9882 tion, or a multi-paragraph argument. Despite this diversity of conditions, all data-to-text
9883 systems share some of the same challenges (Reiter and Dale, 2000):

- 9884 • determining what parts of the data to describe;
- 9885 • planning a presentation of this information;
- 9886 • **lexicalizing** the data into words and phrases;
- 9887 • organizing words and phrases into well-formed sentences and paragraphs.

9888 The earlier stages of this process are sometimes called **content selection** and **text plan-**
9889 **ning**; the later stages are often called **surface realization**.

9890 Early systems for data-to-text generation were modular, with separate software com-
9891 ponents for each task. Artificial intelligence **planning** algorithms can be applied to both

Database:	Temperature			Cloud Sky Cover		
	time	min	mean	max	time	percent (%)
	06:00-21:00	9	15	21	06:00-09:00	25-50
					09:00-12:00	50-75
Wind Speed			Wind Direction			
time			time			
	06:00-21:00			06:00-21:00		
		15	20	30		s

Text: Cloudy, with temperatures between 10 and 20 degrees. South wind around 20 mph.

Figure 19.1: An example input-output pair for the task of generating text descriptions of weather forecasts (Konstas and Lapata, 2013). [todo: permission]

the high-level information structure and the organization of individual sentences, ensuring that communicative goals are met (McKeown, 1992; Moore and Paris, 1993). Surface realization can be performed by grammars or templates, which link specific types of data to candidate words and phrases. A simple example template is offered by Wiseman et al. (2017), for generating descriptions of basketball games:

(19.1) The <team1> (<wins1>-<losses1>) defeated the <team2> (<wins2>-<losses2>),
<pts1>-<pts2>. The New York Knicks (45-5) defeated the Boston Celtics (11-38), 115-79.

For more complex cases, it may be necessary to apply morphological inflections such as pluralization and tense marking — even in the simple example above, languages such as Russian would require case marking suffixes for the team names. Such inflections can be applied as a postprocessing step. Another difficult challenge for surface realization is the generation of varied **referring expressions** (e.g., *The Knicks*, *New York*, *they*), which is critical to avoid repetition. As discussed in § 16.2.1, the form of referring expressions is constrained by the discourse and information structure.

An example at the intersection of rule-based and statistical techniques is the Nitrogen system (Langkilde and Knight, 1998). The input to Nitrogen is an abstract meaning representation (AMR; see § 13.3) of semantic content to be expressed in a single sentence. In data-to-text scenarios, the abstract meaning representation is the output of a higher-level text planning stage. A set of rules then converts the abstract meaning representation into various sentence plans, which may differ in both the high-level structure (e.g., active versus passive voice) as well as the low-level details (e.g., word and phrase choice). Some examples are shown in Figure 19.2. To control the combinatorial explosion in the number of possible realizations for any given meaning, the sentence plans are unified into a single finite-state acceptor, in which word tokens are represented by arcs (see § 9.1.1). A bigram

```
(a / admire-01
  :ARG0 (v / visitor
    :ARG1-of (c / arrive-01
      :ARG4 (j / Japan)))
  :ARG1 (m / "Mount Fuji"))
```

- Visitors who came to Japan admire Mount Fuji.
- Visitors who came in Japan admire Mount Fuji.
- Mount Fuji is admired by the visitor who came in Japan.

Figure 19.2: Abstract meaning representation and candidate surface realizations from the Nitrogen system. Example adapted from Langkilde and Knight (1998).

language model is then used to compute weights on the arcs, so that the shortest path is also the surface realization with the highest bigram language model probability.

More recent systems are unified models that are trained end-to-end using backpropagation. Data-to-text generation shares many properties with machine translation, including a problem of **alignment**: labeled examples provide the data and the text, but they do not specify which parts of the text correspond to which parts of the data. For example, to learn from Figure 19.1, the system must align the word *cloudy* to records in CLOUD SKY COVER, the phrases *10* and *20 degrees* to the MIN and MAX fields in TEMPERATURE, and so on. As in machine translation, both latent variables and neural attention have been proposed as solutions.

19.1.1 Latent data-to-text alignment

Given a dataset of texts and associated records $\{(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$, our goal is to learn a model Ψ , so that

$$\hat{\mathbf{w}} = \underset{\mathbf{w} \in \mathcal{V}^*}{\operatorname{argmax}} \Psi(\mathbf{w}, \mathbf{y}; \theta), \quad [19.1]$$

where \mathcal{V}^* is the set of strings over a discrete vocabulary, and θ is a vector of parameters. The relationship between \mathbf{w} and \mathbf{y} is complex: the data \mathbf{y} may contain dozens of records, and \mathbf{w} may extend to several sentences. To facilitate learning and inference, it would be helpful to decompose the scoring function Ψ into subcomponents. This would be possible if given an **alignment**, specifying which element of \mathbf{y} is expressed in each part of \mathbf{w} (Angeli et al., 2010):

$$\Psi(\mathbf{w}, \mathbf{y}; \theta) = \sum_{m=1}^M \psi_{w,y}(\mathbf{w}_m, \mathbf{y}_{z_m}) + \psi_z(z_m, z_{m-1}), \quad [19.2]$$

where z_m indicates the record aligned to word m . For example, in Figure 19.1, z_1 might specify that the word *cloudy* is aligned to the record *cloud-sky-cover:percent*. The score for this alignment would then be given by the weight on features such as

$$(\textit{cloudy}, \textit{cloud-sky-cover:percent}), \quad [19.3]$$

which could be learned from labeled data $\{(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})\}_{i=1}^N$. The function ψ_z can learn to assign higher scores to alignments that are coherent, referring to the same records in adjacent parts of the text.¹

Several datasets include structured records and natural language text (Barzilay and McKeown, 2005; Chen and Mooney, 2008; Liang and Klein, 2009), but the alignments between text and records are usually not available.² One solution is to model the problem probabilistically, treating the alignment as a latent variable (Liang et al., 2009; Konstas and Lapata, 2013). The model can then be estimated using expectation maximization or sampling (see chapter 5).

19.1.2 Neural data-to-text generation

The **encoder-decoder model** and **neural attention** were introduced in § 18.3 as methods for neural machine translation. They can also be applied to data-to-text generation, with the data acting as the source language (Mei et al., 2016). In neural machine translation, the attention mechanism linked words in the source to words in the target; in data-to-text generation, the attention mechanism can link each part of the generated text back to a record in the data. The biggest departure from translation is in the encoder, which depends on the form of the data.

19.1.2.1 Data encoders

In some types of structured records, all values are drawn from discrete sets. For example, the birthplace of an individual is drawn from a discrete set of possible locations; the diagnosis and treatment of a patient are drawn from an exhaustive list of clinical codes (Johnson et al., 2016). In such cases, vector embeddings can be estimated for each field and possible value: for example, a vector embedding for the field BIRTHPLACE, and another embedding for the value BERKELEY_CALIFORNIA (Bordes et al., 2011). The table of such embeddings serves as the encoding of a structured record (He et al., 2017). It is also possible to compress the entire table into a single vector representation, by **pooling** across the embeddings of each field and value (Lebret et al., 2016).

Sequences Some types of structured records have a natural ordering, such as events in a game (Chen and Mooney, 2008) and steps in a recipe (Tutin and Kittredge, 1992). For example, the following records describe a sequence of events in a robot soccer match (Mei

¹More expressive decompositions of Ψ are possible. For example, Wong and Mooney (2007) use a synchronous context-free grammar (see § 18.2.4) to “translate” between a meaning representation and natural language text.

²An exception is a dataset of records and summaries from American football games, containing annotations of alignments between sentences and records (Snyder and Barzilay, 2007).

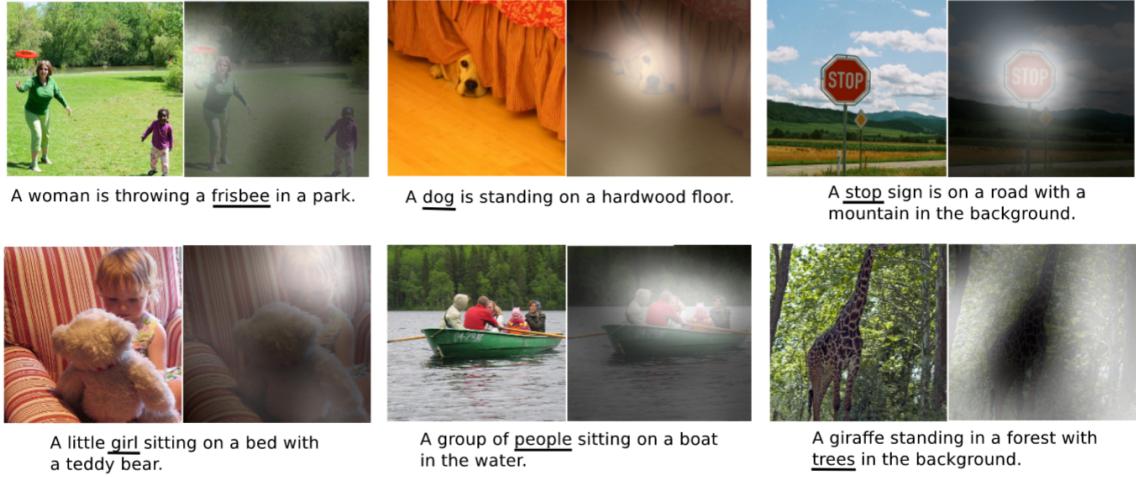


Figure 19.3: Examples of the image captioning task, with attention masks shown for each of the underlined words. From Xu et al. (2015). [todo: permission]

et al., 2016):

```
PASS(arg1 = PURPLE6,arg2 = PURPLE3)
KICK(arg1 = PURPLE3)
BADPASS(arg1 = PURPLE3,arg2 = PINK9).
```

9966 Each event is a single record, and can be encoded by a concatenation of vector representations for the event type (e.g., PASS), the field (e.g., arg1), and the values (e.g., PURPLE3),
9967 e.g.,

$$\mathbf{X} = [\mathbf{u}_{\text{PASS}}, \mathbf{u}_{\text{arg1}}, \mathbf{u}_{\text{PURPLE6}}, \mathbf{u}_{\text{arg2}}, \mathbf{u}_{\text{PURPLE3}}]. \quad [19.4]$$

9969 This encoding can then act as the input layer for a recurrent neural network, yielding a
9970 sequence of vector representations $\{z_r\}_{r=1}^R$, where r indexes over records. Interestingly,
9971 this sequence-based approach is effective even in cases where there is no natural ordering
9972 over the records, such as the weather data in Figure 19.1 (Mei et al., 2016).

9973 **Images** Another flavor of data-to-text generation is the generation of text captions for
9974 images. Examples from this task are shown in Figure 19.3. Images are naturally repre-
9975 sented as tensors: a color image of 320×240 pixels would be stored as a tensor with
9976 $320 \times 240 \times 3$ intensity values. The dominant approach to image classification is to en-
9977 code images as vectors using a combination of convolution and pooling (Krizhevsky et al.,
9978 2012). Chapter 3 explains how to use convolutional networks for text; for images, convo-
9979 lution is applied across the vertical, horizontal, and color dimensions. By pooling the re-
9980 sults of successive convolutions, the image is converted to a vector representation, which

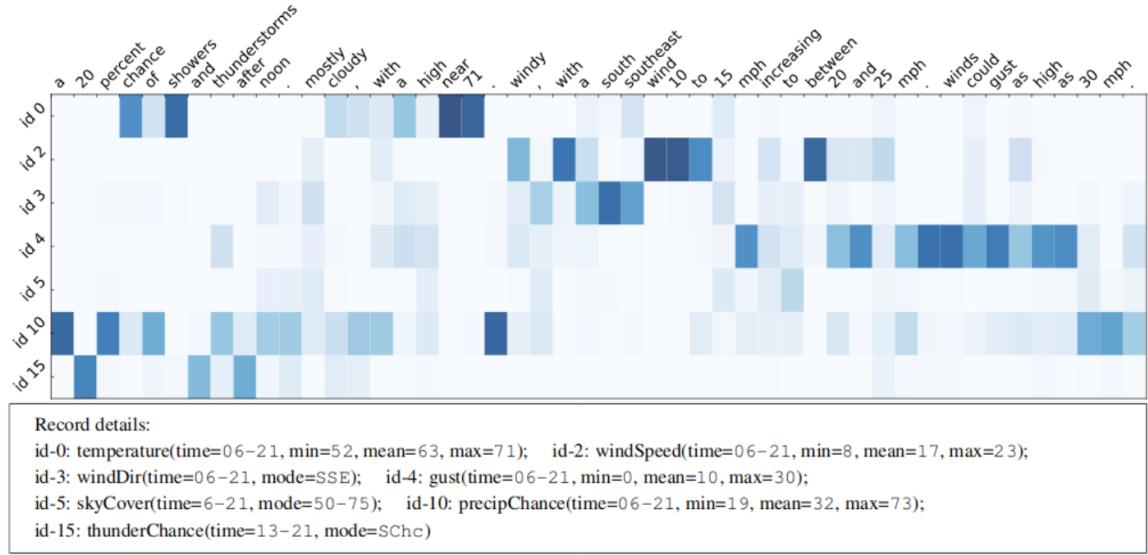


Figure 19.4: Neural attention in text generation. Figure from Mei et al. (2016).[todo: permission]

9981 can then be fed directly into the decoder as the initial state (Vinyals et al., 2015), just as
9982 in the sequence-to-sequence translation model (see § 18.3). Alternatively, one can apply
9983 a set of convolutional networks, yielding vector representations for different parts of the
9984 image, which can then be combined using neural attention (Xu et al., 2015).

9985 19.1.2.2 Attention

Given a set of embeddings of the data $\{\mathbf{z}_r\}_{r=1}^R$ and a decoder state \mathbf{h}_m , the attention vector over the data can be computed using the same technique described in § 18.3.1. When generating word m of the output, a softmax attention mechanism computes the weighted average \mathbf{c}_m ,

$$\psi_\alpha(m, r) = \beta_\alpha \cdot f(\Theta_\alpha[\mathbf{h}_m; \mathbf{z}_r]) \quad [19.5]$$

$$\boldsymbol{\alpha}_m = \text{SoftMax}([\psi_\alpha(m, 1), \psi_\alpha(m, 2), \dots, \psi_\alpha(m, R)]) \quad [19.6]$$

$$\mathbf{c}_m = \sum_{r=1}^R \alpha_{m \rightarrow r} \mathbf{z}_r, \quad [19.7]$$

9986 where f is an elementwise nonlinearity such as tanh or ReLU (see § 3.2.1). The weighted
9987 average \mathbf{c}_m can then be included in the recurrent update to the decoder state, or in the
9988 emission probabilities, as described in § 18.3.1. Figure 19.4 shows the attention to compo-
9989 nents of a weather record, while generating the text shown on the x -axis.

9990 Adapting this architecture to image captioning is straightforward. A convolutional
 9991 neural networks is applied to a set of image locations, and the output at each location ℓ is
 9992 represented with a vector z_ℓ . Attention can then be computed over the image locations,
 9993 as shown in the right panels of each pair of images in Figure 19.3.

9994 Various modifications to this basic mechanism have been proposed. In **coarse-to-fine**
 9995 **attention** (Mei et al., 2016), each record receives a global attention $a_r \in [0, 1]$, which is in-
 9996 dependent of the decoder state. This global attention, which represents the overall impor-
 9997 tance of the record, is multiplied with the decoder-based attention scores, before comput-
 9998 ing the final normalized attentions. In **structured attention**, the attention vector $\alpha_{m \rightarrow \cdot}$ can
 9999 include structural biases, which can favor assigning higher attention values to contiguous
 10000 segments or to dependency subtrees (Kim et al., 2017). Structured attention vectors can
 10001 be computed by running the forward-backward algorithm to obtain marginal attention
 10002 probabilities (see § 7.5.3.3). Because each step in the forward-backward algorithm is dif-
 10003 ferentiable, it can be encoded in a computation graph, and end-to-end learning can be
 10004 performed by backpropagation.

10005 19.1.2.3 Decoder

10006 Given the encoding, the decoder can function just as in neural machine translation (see
 10007 § 18.3.1), using the attention-weighted encoder representation in the decoder recurrence
 10008 and/or output computation. As in machine translation, beam search can help to avoid
 10009 search errors (Lebret et al., 2016).

Many applications require generating words that do not appear in the training vocabulary. For example, a weather record may contain a previously unseen city name; a sports record may contain a previously unseen player name. Such tokens can be generated in the text by copying them over from the input (e.g., Gulcehre et al., 2016).³ First introduce an additional variable $s_m \in \{\text{gen}, \text{copy}\}$, indicating whether token $w_m^{(t)}$ should be generated or copied. The decoder probability is then,

$$p(w^{(t)} | w_{1:m-1}^{(t)}, \mathbf{Z}, s_m) = \begin{cases} \text{SoftMax}(\beta_{w^{(t)}} \cdot h_{m-1}^{(t)}), & s_m = \text{gen} \\ \sum_{r=1}^R \delta(w_r^{(s)} = w^{(t)}) \times \alpha_{m \rightarrow r}, & s_m = \text{copy}, \end{cases} \quad [19.8]$$

10010 where $\delta(w_r^{(s)} = w^{(t)})$ is an indicator function, taking the value 1 iff the text of the record
 10011 $w_r^{(s)}$ is identical to the target word $w^{(t)}$. The probability of copying record r from the source
 10012 is $\delta(s_m = \text{copy}) \times \alpha_{m \rightarrow r}$, the product of the copy probability by the local attention. Note
 10013 that in this model, the attention weights α_m are computed from the *previous* decoder state
 10014 h_{m-1} . The computation graph therefore remains a feedforward network, with recurrent
 10015 paths such as $h_{m-1}^{(t)} \rightarrow \alpha_m \rightarrow w_m^{(t)} \rightarrow h_m^{(t)}$.

³A number of variants of this strategy have been proposed (e.g., Gu et al., 2016; Merity et al., 2017). See Wiseman et al. (2017) for an overview.

10016 To facilitate end-to-end training, the switching variable s_m can be represented by a
 10017 gate π_m , which is computed from a two-layer feedforward network, whose input consists
 10018 of the concatenation of the decoder state $\mathbf{h}_{m-1}^{(t)}$ and the attention-weighted representation
 10019 of the data, $\mathbf{c}_m = \sum_{r=1}^R \alpha_{m \rightarrow r} \mathbf{z}_r$,

$$\pi_m = \sigma(\Theta^{(2)} f(\Theta^{(1)}[\mathbf{h}_{m-1}^{(t)}; \mathbf{c}_m])). \quad [19.9]$$

The full generative probability at token m is then,

$$p(w^{(t)} | \mathbf{w}_{1:m}^{(t)}, \mathbf{Z}) = \pi_m \times \underbrace{\frac{\exp \beta_{w^{(t)}} \cdot \mathbf{h}_{m-1}^{(t)}}{\sum_{j=1}^V \exp \beta_j \cdot \mathbf{h}_{m-1}^{(t)}}}_{\text{generate}} + (1 - \pi_m) \times \underbrace{\sum_{r=1}^R \delta(w_r^{(s)} = w^{(t)}) \times \alpha_{m \rightarrow r}}_{\text{copy}}. \quad [19.10]$$

10020 19.2 Text-to-text generation

10021 Text-to-text generation includes problems of summarization and simplification:

- 10022 • reading a novel and outputting a paragraph-long summary of the plot;⁴
- 10023 • reading a set of blog posts about politics, and outputting a bullet list of the various
 10024 issues and perspectives;
- 10025 • reading a technical research article about the long-term health consequences of drink-
 10026 ing kombucha, and outputting a summary of the article in language that non-experts
 10027 can understand.

10028 These problems can be approached in two ways: through the encoder-decoder architec-
 10029 ture discussed in the previous section, or by operating directly on the input text.

10030 19.2.1 Neural abstractive summarization

10031 **Sentence summarization** is the task of shortening a sentence while preserving its mean-
 10032 ing, as in the following examples (Knight and Marcu, 2000; Rush et al., 2015):

- 10033 (19.2) The documentation is typical of Epson quality: excellent.
 10034 Documentation is excellent.

⁴In § 16.3.4.1, we encountered a special case of single-document summarization, which involved extracting the most important sentences or discourse units. We now consider the more challenging problem of **abstractive summarization**, in which the summary can include words that do not appear in the original text.

10036 (19.3) Russian defense minister Ivanov called sunday for the creation of a joint front for
 10037 combating global terrorism.
 10038 Russia calls for joint front against terrorism.
 10039

10040 Sentence summarization is closely related to **sentence compression**, in which the sum-
 10041 mary is produced by deleting words or phrases from the original (Clarke and Lapata,
 10042 2008). But as shown in (19.3), a sentence summary can also introduce new words, such as
 10043 *against*, which replaces the phrase *for combatting*.

10044 Sentence summarization can be treated as a machine translation problem, using the at-
 10045 tentional encoder-decoder translation model discussed in § 18.3.1 (Rush et al., 2015). The
 10046 longer sentence is encoded into a sequence of vectors, one for each token. The decoder
 10047 then computes attention over these vectors when updating its own recurrent state. As
 10048 with data-to-text generation, it can be useful to augment the encoder-decoder model with
 10049 the ability to copy words directly from the source. Rush et al. (2015) train this model by
 10050 building four million sentence pairs from news articles. In each pair, the longer sentence is
 10051 the first sentence of the article, and the summary is the article headline. Sentence summa-
 10052 rization can also be trained in a semi-supervised fashion, using a probabilistic formulation
 10053 of the encoder-decoder model called a **variational autoencoder** (Miao and Blunsom, 2016,
 10054 also see § 14.8.2).

When summarizing longer documents, an additional concern is that the summary not be repetitive: each part of the summary should cover new ground. This can be addressed by maintaining a vector of the sum total of all attention values thus far, $t_m = \sum_{n=1}^m \alpha_n$. This total can be used as an additional input to the computation of the attention weights,

$$\alpha_{m \rightarrow n} \propto \exp \left(\mathbf{v}_\alpha \cdot \tanh(\Theta_\alpha[\mathbf{h}_m^{(t)}; \mathbf{h}_n^{(s)}; \mathbf{t}_m]) \right), \quad [19.11]$$

which enables the model to learn to prefer parts of the source which have not been attended to yet (Tu et al., 2016). To further encourage diversity in the generated summary, See et al. (2017) introduce a **coverage loss** to the objective function,

$$\ell_m = \sum_{n=1}^{M^{(s)}} \min(\alpha_{m \rightarrow n}, t_{m \rightarrow n}). \quad [19.12]$$

10055 This loss will be low if $\alpha_{m \rightarrow \cdot}$ assigns little attention to words that already have large
 10056 values in $t_{m \rightarrow \cdot}$. Coverage loss is similar to the concept of **marginal relevance**, in which
 10057 the reward for adding new content is proportional to the extent to which it increases
 10058 the overall amount of information conveyed by the summary (Carbonell and Goldstein,
 10059 1998).

10060 **19.2.2 Sentence fusion for multi-document summarization**

10061 In **multi-document summarization**, the goal is to produce a summary that covers the
 10062 content of several documents (McKeown et al., 2002). One approach to this challenging
 10063 problem is to identify sentences across multiple documents that relate to a single theme,
 10064 and then to fuse them into a single sentence (Barzilay and McKeown, 2005). As an exam-
 10065 ple, consider the following two sentences (McKeown et al., 2010):

- 10066 (19.4) Palin actually turned against the bridge project only after it became a national
 10067 symbol of wasteful spending.
 10068 (19.5) Ms. Palin supported the bridge project while running for governor, and aban-
 10069 doned it after it became a national scandal.

10070 An *intersection* preserves only the content that is present in both sentences:

- 10071 (19.6) Palin turned against the bridge project after it became a national scandal.

10072 A *union* includes information from both sentences:

- 10073 (19.7) Ms. Palin supported the bridge project while running for governor, but turned
 10074 against it when it became a national scandal and a symbol of wasteful spending.

Dependency parsing is often used as a technique for sentence fusion. After parsing each sentence, the resulting dependency trees can be aggregated into a lattice (Barzilay and McKeown, 2005) or a graph structure (Filippova and Strube, 2008), in which identical or closely related words (e.g., *Palin*, *bridge*, *national*) are fused into a single node. The resulting graph can then be pruned back to a tree by solving an **integer linear program** (see § 13.2.2),

$$\max_{\mathbf{y}} \sum_{i,j,r} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) \times y_{i,j,r} \quad [19.13]$$

$$\text{s.t. } \mathbf{y} \in \mathcal{C}, \quad [19.14]$$

10075 where the variable $y_{i,j,r} \in \{0, 1\}$ indicates whether there is an edge from i to j of type r ,
 10076 the score of this edge is $\psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})$, and \mathcal{C} is a set of constraints, described below. As
 10077 usual, \mathbf{w} is the list of words in the graph, and $\boldsymbol{\theta}$ is a vector of parameters. The score $\psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})$
 10078 reflects the “importance” of the modifier j to the overall meaning: in intersective
 10079 fusion, this score indicates the extent to which the content in this edge is expressed in all
 10080 sentences; in union fusion, the score indicates whether the content in the edge is expressed
 10081 in any sentence.

10082 The constraint set \mathcal{C} ensures that \mathbf{y} forms a valid dependency graph. It can also im-
 10083 pose additional linguistic constraints: for example, ensuring that coordinated nouns are

10084 sufficiently similar. The resulting tree must then be **linearized** into a sentence. This is
 10085 typically done by generating a set of candidate linearizations, and choosing the one with
 10086 the highest score under a language model (Langkilde and Knight, 1998; Song et al., 2016).

10087 19.3 Dialogue

10088 **Dialogue systems** are capable of conversing with a human interlocutor, often to per-
 10089 form some task (Grosz, 1979), but sometimes just to chat (Weizenbaum, 1966). While re-
 10090 search on dialogue systems goes back several decades (Carbonell, 1970; Winograd, 1972),
 10091 commercial systems such as Alexa and Siri have recently brought this technology into
 10092 widespread use. Nonetheless, there is a significant gap between research and practice:
 10093 many practical dialogue systems remain scripted and inflexible, while research systems
 10094 emphasize abstractive text generation, “on-the-fly” decision making, and probabilistic
 10095 reasoning about the user’s intentions.

10096 19.3.1 Finite-state and agenda-based dialogue systems

10097 Finite-state automata were introduced in chapter 9 as a formal model of computation,
 10098 in which string inputs and outputs are linked to transitions between a finite number of
 10099 discrete states. This model naturally fits simple task-oriented dialogues, such as the one
 10100 shown in the left panel of Figure 19.5. This (somewhat frustrating) dialogue can be repre-
 10101 sented with a finite-state transducer, as shown in the right panel of the figure. The accept-
 10102 ing state is reached only when the two needed pieces of information are provided, and the
 10103 human user confirms that the order is correct. In this simple scenario, the TOPPING and
 10104 ADDRESS are the two **slots** associated with the activity of ordering a pizza, which is called
 10105 a **frame**. Frame representations can be hierarchical: for example, an ADDRESS could have
 10106 slots of its own, such as STREET and CITY.

10107 In the example dialogue in Figure 19.5, the user provides the precise inputs that are
 10108 needed in each turn (e.g., *anchovies*; *the College of Computing building*). Some users may
 10109 prefer to communicate more naturally, with phrases like *I’d, uh, like some anchovies please*.
 10110 One approach to handling such utterances is to design a custom grammar, with non-
 10111 terminals for slots such as TOPPING and LOCATION. However, context-free parsing of
 10112 unconstrained speech input is challenging. A more lightweight alternative is BIO-style
 10113 sequence labeling (see § 8.3), e.g.:

10114 (19.9) *I’d like anchovies , and please bring it to the College of Computing*
 10115 *Building .*
O O B-TOPPING O O O O O O B-ADDR I-ADDR I-ADDR I-ADDR I-ADDR
I-ADDR O

- (19.8) A: I want to order a pizza.
 B: What toppings?
 A: Anchovies.
 B: Ok, what address?
 A: The College of Computing building.
 B: Please confirm: one pizza with artichokes, to be delivered to the College of Computing building.
 A: No.
 B: What toppings?
 ...

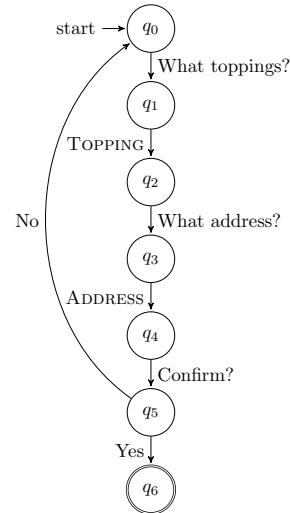


Figure 19.5: An example dialogue and the associated finite-state model. In the finite-state model, SMALL CAPS indicates that the user must provide information of this type in their answer.

10116 The tagger can be driven by a bi-directional recurrent neural network, similar to recurrent
 10117 approaches to semantic role labeling described in § 13.2.3.

10118 The input in (19.9) could not be handled by the finite-state system from Figure 19.5,
 10119 which forces the user to provide the topping first, and then the location. In this sense,
 10120 the **initiative** is driven completely by the system. **Agenda-based dialogue systems** ex-
 10121 tend finite-state architectures by attempting to recognize all slots that are filled by the
 10122 user’s reply, thereby handling these more complex examples. Agenda-based systems dy-
 10123 namically pose additional questions until the frame is complete (Bobrow et al., 1977; Allen
 10124 et al., 1995; Rudnicky and Xu, 1999). Such systems are said to be **mixed-initiative**, because
 10125 both the user and the system can drive the direction of the dialogue.

10126 **19.3.2 Markov decision processes**

10127 The task of dynamically selecting the next move in a conversation is known as **dialogue**
 10128 **management**. This problem can be framed as a **Markov decision process**, which is a
 10129 theoretical model that includes a discrete set of states, a discrete set of actions, a function
 10130 that computes the probability of transitions between states, and a function that computes
 10131 the cost or reward of action-state pairs. Let’s see how each of these elements pertains to
 10132 the pizza ordering dialogue system.

- 10133 • Each state is a tuple of information about whether the topping and address are

10134 known, and whether the order has been confirmed. For example,

(KNOWN TOPPING, UNKNOWN ADDRESS, NOT CONFIRMED) [19.15]

10135 is a possible state. Any state in which the pizza order is confirmed is a terminal
 10136 state, and the Markov decision process stops after entering such a state.

- 10137 • The set of actions includes querying for the topping, querying for the address, and
 10138 requesting confirmation. Each action induces a probability distribution over states,
 10139 $p(s_t | a_t, s_{t-1})$. For example, requesting confirmation of the order is not likely to
 10140 result in a transition to the terminal state if the topping is not yet known. This
 10141 probability distribution over state transitions may be learned from data, or it may
 10142 be specified in advance.
- 10143 • Each state-action-state tuple earns a reward, $r_a(s_t, s_{t+1})$. In the context of the pizza
 10144 ordering system, a simple reward function would be,

$$r_a(s_t, s_{t+1}) = \begin{cases} 0, & a = \text{CONFIRM}, s_{t+1} = (*, *, \text{CONFIRMED}) \\ -10, & a = \text{CONFIRM}, s_{t+1} = (*, *, \text{NOT CONFIRMED}) \\ -1, & a \neq \text{CONFIRM} \end{cases} \quad [19.16]$$

10145 This function assigns zero reward for successful transitions to the terminal state, a
 10146 large negative reward to a rejected request for confirmation, and a small negative re-
 10147 ward for every other type of action. The system is therefore rewarded for reaching
 10148 the terminal state in few steps, and penalized for prematurely requesting confirma-
 10149 tion.

10150 In a Markov decision process, a **policy** is a function $\pi : \mathcal{S} \mapsto \mathcal{A}$ that maps from states to
 10151 actions (see § 15.2.4.3). The value of a policy is the expected sum of discounted rewards,
 10152 $E_\pi[\sum_{t=1}^T \gamma^t r_{a_t}(s_t, s_{t+1})]$, where γ is the discount factor, $\gamma \in [0, 1)$. Discounting has the
 10153 effect of emphasizing rewards that can be obtained immediately over less certain rewards
 10154 in the distant future.

10155 An optimal policy can be obtained by dynamic programming, by iteratively updating
 10156 the **value function** $V(s)$, which is the expectation of the cumulative reward from s under
 10157 the optimal action a ,

$$V(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s' | s, a)[r_a(s, s') + \gamma V(s')]. \quad [19.17]$$

10158 The value function $V(s)$ is computed in terms of $V(s')$ for all states $s' \in \mathcal{S}$. A series
 10159 of iterative updates to the value function will eventually converge to a stationary point.
 10160 This algorithm is known as **value iteration**. Given the converged value function $V(s)$, the

10161 optimal action at each state is the argmax,

$$\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s' | s, a)[r_a(s, s') + \gamma V(s')]. \quad [19.18]$$

10162 Value iteration and related algorithms are described in detail by Sutton and Barto (1998).
 10163 For applications to dialogue systems, see Levin et al. (1998) and Walker (2000).

10164 The Markov decision process framework assumes that the current state of the dialogue
 10165 is known. In reality, the system may misinterpret the user’s statements — for example,
 10166 believing that a specification of the delivery location (PEACHTREE) is in fact a specification
 10167 of the topping (PEACHES). In a **partially observable Markov decision process (POMDP)**,
 10168 the system receives an *observation* o , which is probabilistically conditioned on the state,
 10169 $p(o | s)$. It must therefore maintain a distribution of beliefs about which state it is in, with
 10170 $q_t(s)$ indicating the degree of belief that the dialogue is in state s at time t . The POMDP
 10171 formulation can help to make dialogue systems more robust to errors, particularly in the
 10172 context of spoken language dialogues, where the speech itself may be misrecognized (Roy
 10173 et al., 2000; Williams and Young, 2007). However, finding the optimal policy in a POMDP
 10174 is computationally intractable, requiring additional approximations.

10175 19.3.3 Neural chatbots

10176 Chatting is a lot easier when you don’t need to get anything done. **Chatbots** are systems
 10177 that parry the user’s input with a response that keeps the conversation going. They can be
 10178 built from the encoder-decoder architecture discussed in § 18.3 and § 19.1.2: the encoder
 10179 converts the user’s input into a vector, and the decoder produces a sequence of words as a
 10180 response. For example, Shang et al. (2015) apply the attentional encoder-decoder transla-
 10181 tion model, training on a dataset of posts and responses from the Chinese microblogging
 10182 platform Sina Weibo.⁵ This approach is capable of generating replies that relate themati-
 10183 cally to the input, as shown in the following examples:⁶

10184 (19.10) A: High fever attacks me every New Year’s day.
 10185 Get B: well soon and stay healthy!

10186 (19.11) A: I gain one more year. Grateful to my group, so happy.
 10187 B: Getting old now. Time has no mercy.

10188 While encoder-decoder models can generate responses that make sense in the con-
 10189 text of the immediately preceding turn, they struggle to maintain coherence over longer

⁵Twitter is also frequently used for construction of dialogue datasets (Ritter et al., 2011; Sordoni et al., 2015). Another source is technical support chat logs from the Ubuntu linux distribution (Uthus and Aha, 2013; Lowe et al., 2015).

⁶All examples are translated from Chinese by Shang et al. (2015).

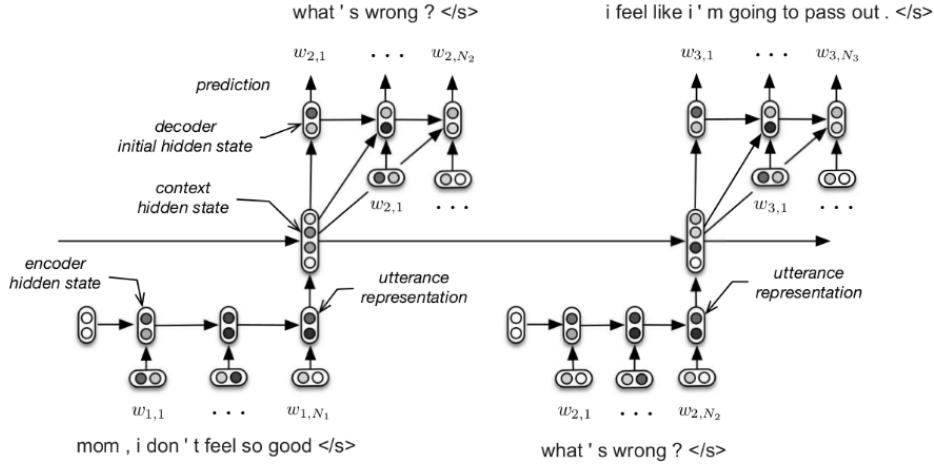


Figure 19.6: A hierarchical recurrent neural network for dialogue, with recurrence over both words and turns, from Serban et al. (2016). [todo: permission]

conversations. One solution is to model the dialogue context recurrently. This creates a **hierarchical recurrent network**, including both word-level and turn-level recurrences. The turn-level hidden state is then used as additional context in the decoder (Serban et al., 2016), as shown in Figure 19.6.

An open question is how to integrate the encoder-decoder architecture into task-oriented dialogue systems. Neural chatbots can be trained end-to-end: the user's turn is analyzed by the encoder, and the system output is generated by the decoder. This architecture can be trained by log-likelihood using backpropagation (e.g., Sordoni et al., 2015; Serban et al., 2016), or by more elaborate objectives, using reinforcement learning (Li et al., 2016). In contrast, the task-oriented dialogue systems described in § 19.3.1 typically involve a set of specialized modules: one for recognizing the user input, another for deciding what action to take, and a third for arranging the text of the system output.

Recurrent neural network decoders can be integrated into Markov Decision Process dialogue systems, by conditioning the decoder on a representation of the information that is to be expressed in each turn (Wen et al., 2015). Specifically, the long short-term memory (LSTM; § 6.3) architecture is augmented so that the memory cell at turn m takes an additional input d_m , which is a representation of the slots and values to be expressed in the next turn. However, this approach still relies on additional modules to recognize the user's utterance and to plan the overall arc of the dialogue.

Another promising direction is to create embeddings for the elements in the domain: for example, the slots in a record and the entities that can fill them. The encoder then

10211 encodes not only the words of the user’s input, but the embeddings of the elements that
 10212 the user mentions. Similarly, the decoder is endowed with the ability to refer to specific
 10213 elements in the knowledge base. He et al. (2017) show that such a method can learn to
 10214 play a collaborative dialogue game, in which both players are given a list of entities and
 10215 their properties, and the goal is to find an entity that is on both players’ lists.

10216 **Further reading**

10217 Gatt and Krahmer (2018) provide a comprehensive recent survey on text generation. For
 10218 a book-length treatment of earlier work, see Reiter and Dale (2000). For a survey on image
 10219 captioning, see Bernardi et al. (2016); for a survey of pre-neural approaches to dialogue
 10220 systems, see Rieser and Lemon (2011). **Dialogue acts** were introduced in § 8.6 as a labeling
 10221 scheme for human-human dialogues; they also play a critical role in task-based dialogue
 10222 systems (e.g., Allen et al., 1996). The incorporation of theoretical models of dialogue into
 10223 computational systems is reviewed by Jurafsky and Martin (2009, chapter 24).

10224 While this chapter has focused on the informative dimension of text generation, another
 10225 line of research aims to generate text with configurable stylistic properties (Walker
 10226 et al., 1997; Mairesse and Walker, 2011; Ficler and Goldberg, 2017; Hu et al., 2017). This
 10227 chapter also does not address the generation of creative text such as narratives (Riedl and
 10228 Young, 2010), jokes (Ritchie, 2001), poems (Colton et al., 2012), and song lyrics (Gonçalo Oliveira
 10229 et al., 2007).

10230 **Exercises**

10231 1. The SimpleNLG system produces surface realizations from representations of de-
 10232 sired syntactic structure (Gatt and Reiter, 2009). This system can be accessed on
 10233 github at <https://github.com/simpleNLG/simpleNLG>. Download the sys-
 10234 tem, and produce realizations of the following examples:

- 10235 (19.12) Call me Ismael.
 10236 (19.13) I try all things.
 10237 (19.14) I achieve what I can.

10238 Then convert each example to a question. [todo: Can’t get SimpleNLG to work with
 10239 python anymore]

10240 **Appendix A**

10241 **Probability**

10242 Probability theory provides a way to reason about random events. The sorts of random
10243 events that are typically used to explain probability theory include coin flips, card draws,
10244 and the weather. It may seem odd to think about the choice of a word as akin to the flip of
10245 a coin, particularly if you are the type of person to choose words carefully. But random or
10246 not, language has proven to be extremely difficult to model deterministically. Probability
10247 offers a powerful tool for modeling and manipulating linguistic data.

10248 Probability can be thought of in terms of **random outcomes**: for example, a single coin
10249 flip has two possible outcomes, heads or tails. The set of possible outcomes is the **sample**
10250 **space**, and a subset of the **sample space** is an **event**. For a sequence of two coin flips,
10251 there are four possible outcomes, $\{HH, HT, TH, TT\}$, representing the ordered sequences
10252 heads-head, heads-tails, tails-heads, and tails-tails. The event of getting exactly one head
10253 includes two outcomes: $\{HT, TH\}$.

10254 Formally, a probability is a function from events to the interval between zero and one:
10255 $\Pr : \mathcal{F} \mapsto [0, 1]$, where \mathcal{F} is the set of possible events. An event that is certain has proba-
10256 bility one; an event that is impossible has probability zero. For example, the probability
10257 of getting fewer than three heads on two coin flips is one. Each outcome is also an event
10258 (a set with exactly one element), and for two flips of a fair coin, the probability of each
10259 outcome is,

$$\Pr(\{HH\}) = \Pr(\{HT\}) = \Pr(\{TH\}) = \Pr(\{TT\}) = \frac{1}{4}. \quad [A.1]$$

10260 **A.1 Probabilities of event combinations**

10261 Because events are sets of outcomes, we can use set-theoretic operations such as comple-
10262 ment, intersection, and union to reason about the probabilities of events and their combi-
10263 nations.

10264 For any event A , there is a **complement** $\neg A$, such that:

10265 • The probability of the union $A \cup \neg A$ is $\Pr(A \cup \neg A) = 1$;

10266 • The intersection $A \cap \neg A = \emptyset$ is the empty set, and $\Pr(A \cap \neg A) = 0$.

10267 In the coin flip example, the event of obtaining a single head on two flips corresponds to
10268 the set of outcomes $\{HT, TH\}$; the complement event includes the other two outcomes,
10269 $\{TT, HH\}$.

10270 A.1.1 Probabilities of disjoint events

10271 When two events have an empty intersection, $A \cap B = \emptyset$, they **disjoint**. The probability
10272 of the union of two disjoint events is equal to the sum of their probabilities,

$$A \cap B = \emptyset \Rightarrow \Pr(A \cup B) = \Pr(A) + \Pr(B). \quad [A.2]$$

10273 This is the **third axiom of probability**, and it can be generalized to any countable sequence
10274 of disjoint events.

In the coin flip example, this axiom can derive the probability of the event of getting a single head on two flips. This event is the set of outcomes $\{HT, TH\}$, which is the union of two simpler events, $\{HT, TH\} = \{HT\} \cup \{TH\}$. The events $\{HT\}$ and $\{TH\}$ are disjoint. Therefore,

$$\Pr(\{HT, TH\}) = \Pr(\{HT\} \cup \{TH\}) = \Pr(\{HT\}) + \Pr(\{TH\}) \quad [A.3]$$

$$= \frac{1}{4} + \frac{1}{4} = \frac{1}{2}. \quad [A.4]$$

10275 In the general, the probability of the union of two events is,

$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B). \quad [A.5]$$

This can be seen visually in Figure A.1, and it can be derived from the third axiom of probability. Consider an event that includes all outcomes in B that are not in A , denoted as $B - (A \cap B)$. By construction, this event is disjoint from A . We can therefore apply the additive rule,

$$\Pr(A \cup B) = \Pr(A) + \Pr(B - (A \cap B)). \quad [A.6]$$

Furthermore, the event B is the union of two disjoint events: $A \cap B$ and $B - (A \cap B)$.

$$\Pr(B) = \Pr(B - (A \cap B)) + \Pr(A \cap B). \quad [A.7]$$

Reorganizing and substituting into Equation A.6 gives the desired result:

$$\Pr(B - (A \cap B)) = \Pr(B) - \Pr(A \cap B) \quad [A.8]$$

$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B). \quad [A.9]$$

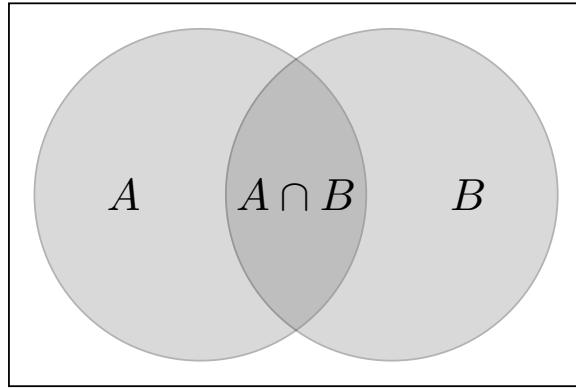


Figure A.1: A visualization of the probability of non-disjoint events A and B .

10276 A.1.2 Law of total probability

10277 A set of events $\mathcal{B} = \{B_1, B_2, \dots, B_N\}$ is a **partition** of the sample space iff each pair of
 10278 events is disjoint ($B_i \cap B_j = \emptyset$), and the union of the events is the entire sample space.
 10279 The law of total probability states that we can **marginalize** over these events as follows,

$$\Pr(A) = \sum_{B_n \in \mathcal{B}} \Pr(A \cap B_n). \quad [\text{A.10}]$$

10280 For any event B , the union $B \cup \neg B$ is a partition of the sample space. Therefore, a special
 10281 case of the law of total probability is,

$$\Pr(A) = \Pr(A \cap B) + \Pr(A \cap \neg B). \quad [\text{A.11}]$$

10282 A.2 Conditional probability and Bayes' rule

A **conditional probability** is an expression like $\Pr(A \mid B)$, which is the probability of the event A , assuming that event B happens too. For example, we may be interested in the probability of a randomly selected person answering the phone by saying *hello*, conditioned on that person being a speaker of English. Conditional probability is defined as the ratio,

$$\Pr(A \mid B) = \frac{\Pr(A \cap B)}{\Pr(B)}. \quad [\text{A.12}]$$

The **chain rule of probability** states that $\Pr(A \cap B) = \Pr(A \mid B) \times \Pr(B)$, which is just

Jacob Eisenstein. Draft of June 11, 2018. Copyright MIT Press, CC-BY-NC-ND license.

a rearrangement of terms from Equation A.12. The chain rule can be applied repeatedly:

$$\begin{aligned}\Pr(A \cap B \cap C) &= \Pr(A | B \cap C) \times \Pr(B \cap C) \\ &= \Pr(A | B \cap C) \times \Pr(B | C) \times \Pr(C).\end{aligned}$$

Bayes' rule (sometimes called Bayes' law or Bayes' theorem) gives us a way to convert between $\Pr(A | B)$ and $\Pr(B | A)$. It follows from the definition of conditional probability and the chain rule:

$$\Pr(A | B) = \frac{\Pr(A \cap B)}{\Pr(B)} = \frac{\Pr(B | A) \times \Pr(A)}{\Pr(B)} \quad [\text{A.13}]$$

10283 Each term in Bayes rule has a name, which we will occasionally use:

- 10284 • Pr(A) is the **prior**, since it is the probability of event A without knowledge about
10285 whether B happens or not.
- 10286 • Pr($B | A$) is the **likelihood**, the probability of event B given that event A has oc-
10287 curred.
- 10288 • Pr($A | B$) is the **posterior**, the probability of event A with knowledge that B has
10289 occurred.

10290 **Example** The classic examples for Bayes' rule involve tests for rare diseases, but Man-
10291 ning and Schütze (1999) reframe this example in a linguistic setting. Suppose that you are
10292 interested in a rare syntactic construction, such as *parasitic gaps*, which occur on average
10293 once in 100,000 sentences. Here is an example of a parasitic gap:

10294 (A.1) *Which class did you attend ... without registering for ...?*

10295 Lana Linguist has developed a complicated pattern matcher that attempts to identify
10296 sentences with parasitic gaps. It's pretty good, but it's not perfect:

- 10297 • If a sentence has a parasitic gap, the pattern matcher will find it with probability
10298 0.95. (This is the **recall**, which is one minus the **false positive rate**.)
- 10299 • If the sentence doesn't have a parasitic gap, the pattern matcher will wrongly say it
10300 does with probability 0.005. (This is the **false positive rate**, which is one minus the
10301 **precision**.)

10302 Suppose that Lana's pattern matcher says that a sentence contains a parasitic gap. What
10303 is the probability that this is true?

Let G be the event of a sentence having a parasitic gap, and T be the event of the test being positive. We are interested in the probability of a sentence having a parasitic gap given that the test is positive. This is the conditional probability $\Pr(G | T)$, and it can be computed by Bayes' rule:

$$\Pr(G | T) = \frac{\Pr(T | G) \times \Pr(G)}{\Pr(T)}. \quad [\text{A.14}]$$

10304 We already know both terms in the numerator: $\Pr(T | G)$ is the recall, which is 0.95; $\Pr(G)$
10305 is the prior, which is 10^{-5} .

10306 We are not given the denominator, but it can be computed using tools developed earlier
10307 in this section. First apply the law of total probability, using the partition $\{G, \neg G\}$:

$$\Pr(T) = \Pr(T \cap G) + \Pr(T \cap \neg G). \quad [\text{A.15}]$$

This says that the probability of the test being positive is the sum of the probability of a **true positive** ($T \cap G$) and the probability of a **false positive** ($T \cap \neg G$). The probability of each of these events can be computed using the chain rule:

$$\Pr(T \cap G) = \Pr(T | G) \times \Pr(G) = 0.95 \times 10^{-5} \quad [\text{A.16}]$$

$$\Pr(T \cap \neg G) = \Pr(T | \neg G) \times \Pr(\neg G) = 0.005 \times (1 - 10^{-5}) \approx 0.005 \quad [\text{A.17}]$$

$$\Pr(T) = \Pr(T \cap G) + \Pr(T \cap \neg G) \quad [\text{A.18}]$$

$$= 0.95 \times 10^{-5} + 0.005. \quad [\text{A.19}]$$

Plugging these terms into Bayes' rule gives the desired posterior probability,

$$\Pr(G | T) = \frac{\Pr(T | G) \Pr(G)}{\Pr(T)} \quad [\text{A.20}]$$

$$= \frac{0.95 \times 10^{-5}}{0.95 \times 10^{-5} + 0.005 \times (1 - 10^{-5})} \quad [\text{A.21}]$$

$$\approx 0.002. \quad [\text{A.22}]$$

10308 Lana's pattern matcher seems accurate, with false positive and false negative rates
10309 below 5%. Yet the extreme rarity of the phenomenon means that a positive result from the
10310 detector is most likely to be wrong.

10311 A.3 Independence

Two events are independent if the probability of their intersection is equal to the product of their probabilities: $\Pr(A \cap B) = \Pr(A) \times \Pr(B)$. For example, for two flips of a fair

coin, the probability of getting heads on the first flip is independent of the probability of getting heads on the second flip:

$$\Pr(\{HT, HH\}) = \Pr(HT) + \Pr(HH) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2} \quad [A.23]$$

$$\Pr(\{HH, TH\}) = \Pr(HH) + \Pr(TH) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2} \quad [A.24]$$

$$\Pr(\{HT, HH\}) \times \Pr(\{HH, TH\}) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4} \quad [A.25]$$

$$\Pr(\{HT, HH\} \cap \{HH, TH\}) = \Pr(HH) = \frac{1}{4} \quad [A.26]$$

$$= \Pr(\{HT, HH\}) \times \Pr(\{HH, TH\}). \quad [A.27]$$

If $\Pr(A \cap B \mid C) = \Pr(A \mid C) \times \Pr(B \mid C)$, then the events A and B are **conditionally independent**, written $A \perp B \mid C$. Conditional independence plays a important role in probabilistic models such as Naïve Bayes chapter 2.

A.4 Random variables

Random variables are functions from events to \mathbb{R}^n , where \mathbb{R} is the set of real numbers. This subsumes several useful special cases:

- An **indicator random variable** is a functions from events to the set $\{0, 1\}$. In the coin flip example, we can define Y as an indicator random variable, taking the value 1 when the coin has come up heads on at least one flip. This would include the outcomes $\{HH, HT, TH\}$. The probability $\Pr(Y = 1)$ is the sum of the probabilities of these outcomes, $\Pr(Y = 1) = \frac{1}{4} + \frac{1}{4} + \frac{1}{4} = \frac{3}{4}$.
- A **discrete random variable** is a function from events to a discrete subset of \mathbb{R} . Consider the coin flip example: the number of heads on two flips, X , can be viewed as a discrete random variable, $X \in \{0, 1, 2\}$. The event probability $\Pr(X = 1)$ can again be computed as the sum of the probabilities of the events in which there is one head, $\{HT, TH\}$, giving $\Pr(X = 1) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$.

Each possible value of a random variable is associated with a subset of the sample space. In the coin flip example, $X = 0$ is associated with the event $\{TT\}$, $X = 1$ is associated with the event $\{HT, TH\}$, and $X = 2$ is associated with the event $\{HH\}$. Assuming a fair coin, the probabilities of these events are, respectively, $1/4$, $1/2$, and $1/4$. This list of numbers represents the **probability distribution** over X , written p_X , which maps from the possible values of X to the non-negative reals. For a specific value x , we write $p_X(x)$, which is equal to the event probability $\Pr(X = x)$.¹ The function p_X is called

¹In general, capital letters (e.g., X) refer to random variables, and lower-case letters (e.g., x) refer to specific values. When the distribution is clear from context, I will simply write $p(x)$.

a probability **mass** function (pmf) if X is discrete; it is called a probability **density** function (pdf) if X is continuous. In either case, the function must sum to one, and all values must be non-negative:

$$\int_x p_X(x)dx = 1 \quad [A.28]$$

$$\forall x, p_X(x) \geq 0. \quad [A.29]$$

Probabilities over multiple random variables can written as **joint probabilities**, e.g., $p_{A,B}(a,b) = \Pr(A = a \cap B = b)$. Several properties of event probabilities carry over to probability distributions over random variables:

- The **marginal probability distribution** is $p_A(a) = \sum_b p_{A,B}(a,b)$.
- The **conditional probability distribution** is $p_{A|B}(a | b) = \frac{p_{A,B}(a,b)}{p_B(b)}$.
- Random variables A and B are independent iff $p_{A,B}(a,b) = p_A(a) \times p_B(b)$.

A.5 Expectations

Sometimes we want the **expectation** of a function, such as $E[g(x)] = \sum_{x \in \mathcal{X}} g(x)p(x)$. Expectations are easiest to think about in terms of probability distributions over discrete events:

- If it is sunny, Lucia will eat three ice creams.
- If it is rainy, she will eat only one ice cream.
- There's a 80% chance it will be sunny.
- The expected number of ice creams she will eat is $0.8 \times 3 + 0.2 \times 1 = 2.6$.

If the random variable X is continuous, the expectation is an integral:

$$E[g(x)] = \int_{\mathcal{X}} g(x)p(x)dx \quad [A.30]$$

For example, a fast food restaurant in Quebec has a special offer for cold days: they give a 1% discount on poutine for every degree below zero. Assuming a thermometer with infinite precision, the expected price would be an integral over all possible temperatures,

$$E[\text{price}(x)] = \int_{\mathcal{X}} \min(1, 1+x) \times \text{original-price} \times p(x)dx. \quad [A.31]$$

10346 **A.6 Modeling and estimation**

10347 **Probabilistic models** provide a principled way to reason about random events and ran-
10348 dom variables. Let's consider the coin toss example. Each toss can be modeled as a ran-
10349 dom event, with probability θ of the event H , and probability $1 - \theta$ of the complementary
10350 event T . If we write a random variable X as the total number of heads on three coin
10351 flips, then the distribution of X depends on θ . In this case, X is distributed as a **binomial**
10352 **random variable**, meaning that it is drawn from a binomial distribution, with **parameters**
10353 $(\theta, N = 3)$. This is written,

$$X \sim \text{Binomial}(\theta, N = 3). \quad [\text{A.32}]$$

10354 The properties of the binomial distribution enable us to make statements about the X ,
10355 such as its expected value and the likelihood that its value will fall within some interval.

Now suppose that θ is unknown, but we have run an experiment, in which we exe-
 cuted N trials, and obtained x heads. We can **estimate** θ by the principle of **maximum**
likelihood:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} p_X(x; \theta, N). \quad [\text{A.33}]$$

This says that the estimate $\hat{\theta}$ should be the value that maximizes the likelihood of the
 data. The semicolon indicates that θ and N are parameters of the probability function.
 The likelihood $p_X(x; \theta, N)$ can be computed from the binomial distribution,

$$p_X(x; \theta, N) = \frac{N!}{x!(N-x)!} \theta^x (1-\theta)^{N-x}. \quad [\text{A.34}]$$

10356 This likelihood is proportional to the product of the probability of individual out-
10357 comes: for example, the sequence T, H, H, T, H would have probability $\theta^3(1-\theta)^2$. The
10358 term $\frac{N!}{x!(N-x)!}$ arises from the many possible orderings by which we could obtain x heads
10359 on N trials. This term does not depend on θ , so it can be ignored during estimation.

In practice, we maximize the log-likelihood, which is a monotonic function of the like-
 lihood. Under the binomial distribution, the log-likelihood is a **convex** function of θ (see

§ 2.3), so it can be maximized by taking the derivative and setting it equal to zero.

$$\ell(\theta) = x \log \theta + (N - x) \log(1 - \theta) \quad [\text{A.35}]$$

$$\frac{\partial \ell(\theta)}{\partial \theta} = \frac{x}{\theta} - \frac{N - x}{1 - \theta} \quad [\text{A.36}]$$

$$\frac{N - x}{1 - \theta} = \frac{x}{\theta} \quad [\text{A.37}]$$

$$\frac{N - x}{x} = \frac{1 - \theta}{\theta} \quad [\text{A.38}]$$

$$\frac{N}{x} - 1 = \frac{1}{\theta} - 1 \quad [\text{A.39}]$$

$$\hat{\theta} = \frac{x}{N}. \quad [\text{A.40}]$$

10360 In this case, the maximum likelihood estimate is equal to $\frac{x}{N}$, the fraction of trials that
 10361 came up heads. This intuitive solution is also known as the **relative frequency estimate**,
 10362 since it is equal to the relative frequency of the outcome.

Is maximum likelihood estimation always the right choice? Suppose you conduct one trial, and get heads. Would you conclude that $\theta = 1$, meaning that the coin is guaranteed to come up heads? If not, then you must have some **prior expectation** about θ . To incorporate this prior information, we can treat θ as a random variable, and use Bayes' rule:

$$p(\theta | x; N) = \frac{p(x | \theta) \times p(\theta)}{p(x)} \quad [\text{A.41}]$$

$$\propto p(x | \theta) \times p(\theta) \quad [\text{A.42}]$$

$$\hat{\theta} = \operatorname{argmax}_{\theta} p(x | \theta) \times p(\theta). \quad [\text{A.43}]$$

10363 This is the **maximum a posteriori** (MAP) estimate. Given a form for $p(\theta)$, you can de-
 10364 rive the MAP estimate using the same approach that was used to derive the maximum
 10365 likelihood estimate.

10366 Additional resources

10367 A good introduction to probability theory is offered by Manning and Schütze (1999),
 10368 which helped to motivate this section. For more detail, Sharon Goldwater provides an-
 10369 other useful reference, <http://homepages.inf.ed.ac.uk/sgwater/teaching/general/probability.pdf>. A historical and philosophical perspective on probability is offered
 10370 by Diaconis and Skyrms (2017).

10372 **Appendix B**

10373 **Numerical optimization**

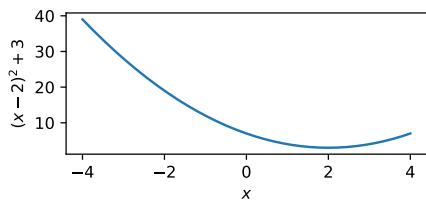
10374 Unconstrained numerical optimization involves solving problems of the form,

$$\min_{\mathbf{x} \in \mathbb{R}^D} f(\mathbf{x}), \quad [\text{B.1}]$$

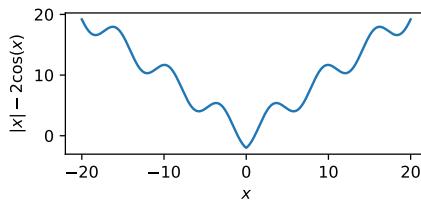
10375 where $\mathbf{x} \in \mathbb{R}^D$ is a vector of D real numbers.

10376 Differentiation is fundamental to continuous optimization. Suppose that at some \mathbf{x}^* ,
10377 every partial derivative is equal to 0: formally, $\frac{\partial f}{\partial x_i}\Big|_{\mathbf{x}^*} = 0$. Then \mathbf{x}^* is said to be a **critical**
10378 **point** of f . For a **convex** function f (defined in § 2.3), $f(\mathbf{x}^*)$ is equal to the global minimum
10379 of f iff \mathbf{x}^* is a critical point of f .

As an example, consider the convex function $f(x) = (x - 2)^2 + 3$, shown in Figure B.1a. The derivative is $\frac{\partial f}{\partial x} = 2x - 4$. A unique minimum can be obtained by setting the derivative equal to zero and solving for x , obtaining $x^* = 2$. Now consider the multivariate convex function $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{x} - [2, 1]^\top\|^2$, where $\|\mathbf{x}\|^2$ is the squared Euclidean norm. The partial



(a) The function $f(x) = (x - 2)^2 + 3$



(b) The function $f(x) = |x| - 2\cos(x)$

Figure B.1: Two functions with unique global minima

derivatives are,

$$\frac{\partial d}{\partial x_1} = x_1 - 2 \quad [B.2]$$

$$\frac{\partial d}{\partial x_2} = x_2 - 1 \quad [B.3]$$

10380 The unique minimum is $\mathbf{x}^* = [2, 1]^\top$.

10381 For non-convex functions, critical points are not necessarily global minima. A **local**
 10382 **minimum** \mathbf{x}^* is a point at which the function takes a smaller value than at all nearby
 10383 neighbors: formally, \mathbf{x}^* is a local minimum if there is some positive ϵ such that $f(\mathbf{x}^*) \leq$
 10384 $f(\mathbf{x})$ for all \mathbf{x} within distance ϵ of \mathbf{x}^* . Figure B.1b shows the function $f(x) = |x| - 2 \cos(x)$,
 10385 which has many local minima, as well as a unique global minimum at $x = 0$. A critical
 10386 point may also be the local or global maximum of the function; it may be a **saddle point**,
 10387 which is a minimum with respect to at least one coordinate, and a maximum with respect
 10388 to at least one other coordinate; it may be an **inflection point**, which is neither a minimum
 10389 nor maximum. When available, the second derivative of f can help to distinguish these
 10390 cases.

10391 B.1 Gradient descent

For many convex functions, it is not possible to solve for \mathbf{x}^* in closed form. In gradient descent, we compute a series of solutions, $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \dots$ by taking steps along the local gradient $\nabla_{\mathbf{x}^{(t)}} f$, which is the vector of partial derivatives of the function f , evaluated at the point $\mathbf{x}^{(t)}$. Each solution $\mathbf{x}^{(t+1)}$ is computed,

$$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} - \eta^{(t)} \nabla_{\mathbf{x}^{(t)}} f. \quad [B.4]$$

10392 where $\eta^{(t)} > 0$ is a **step size**. If the step size is chosen appropriately, this procedure will
 10393 find the global minimum of a differentiable convex function. For non-convex functions,
 10394 gradient descent will find a local minimum. The extension to non-differentiable convex
 10395 functions is discussed in § 2.3.

10396 B.2 Constrained optimization

Optimization must often be performed under constraints: for example, when optimizing the parameters of a probability distribution, the probabilities of all events must sum to one. Constrained optimization problems can be written,

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad [B.5]$$

$$\text{s.t. } g_c(\mathbf{x}) \leq 0, \quad \forall c = 1, 2, \dots, C \quad [B.6]$$

where each $g_i(\mathbf{x})$ is a scalar function of \mathbf{x} . For example, suppose that \mathbf{x} must be non-negative, and that its sum cannot exceed a budget b . Then there are $D + 1$ inequality constraints,

$$g_i(\mathbf{x}) = -x_i, \quad \forall i = 1, 2, \dots, D \quad [\text{B.7}]$$

$$g_{D+1}(\mathbf{x}) = -b + \sum_{i=1}^D x_i. \quad [\text{B.8}]$$

Inequality constraints can be combined with the original objective function f by forming a **Lagrangian**,

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{c=1}^C \lambda_c g_c(\mathbf{x}), \quad [\text{B.9}]$$

where λ_c is a **Lagrange multiplier**. For any Lagrangian, there is a corresponding **dual form**, which is a function of $\boldsymbol{\lambda}$:

$$D(\boldsymbol{\lambda}) = \min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}). \quad [\text{B.10}]$$

The Lagrangian L can be referred to as the **primal form**.

B.3 Example: Passive-aggressive online learning

Sometimes it is possible to solve a constrained optimization problem by manipulating the Lagrangian. One example is maximum-likelihood estimation of a Naïve Bayes probability model, as described in § 2.1.3. In that case, it is unnecessary to explicitly compute the Lagrange multiplier. Another example is illustrated by the **passive-aggressive** algorithm for online learning (Crammer et al., 2006). This algorithm is similar to the perceptron, but the goal at each step is to make the most conservative update that gives zero margin loss on the current example.¹ Each update can be formulated as a constrained optimization over the weights $\boldsymbol{\theta}$:

$$\min_{\boldsymbol{\theta}} \frac{1}{2} \|\boldsymbol{\theta} - \boldsymbol{\theta}^{(i-1)}\|^2 \quad [\text{B.11}]$$

$$\text{s.t. } \ell^{(i)}(\boldsymbol{\theta}) = 0 \quad [\text{B.12}]$$

where $\boldsymbol{\theta}^{(i-1)}$ is the previous set of weights, and $\ell^{(i)}(\boldsymbol{\theta})$ is the margin loss on instance i . As in § 2.3.1, this loss is defined as,

$$\ell^{(i)}(\boldsymbol{\theta}) = 1 - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) + \max_{y \neq y^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y). \quad [\text{B.13}]$$

¹This is the basis for the name of the algorithm: it is passive when the loss is zero, but it aggressively moves to make the loss zero when necessary.

When the margin loss is zero for $\theta^{(i-1)}$, the optimal solution is simply to set $\theta^* = \theta^{(i-1)}$, so we will focus on the case where $\ell^{(i)}(\theta^{(i-1)}) > 0$. The Lagrangian for this problem is,

$$L(\theta, \lambda) = \frac{1}{2} \|\theta - \theta^{(i-1)}\|^2 + \lambda \ell^{(i)}(\theta), \quad [\text{B.14}]$$

Holding λ constant, we can solve for θ by differentiating,

$$\nabla_{\theta} L = \theta - \theta^{(i-1)} + \lambda \frac{\partial}{\partial \theta} \ell^{(i)}(\theta) \quad [\text{B.15}]$$

$$\theta^* = \theta^{(i-1)} + \lambda \delta, \quad [\text{B.16}]$$

where $\delta = f(x^{(i)}, y^{(i)}) - f(x^{(i)}, \hat{y})$ and $\hat{y} = \operatorname{argmax}_{y \neq y^{(i)}} \theta \cdot f(x^{(i)}, y)$.

The Lagrange multiplier λ acts as the learning rate in a perceptron-style update to θ . We can solve for λ by plugging θ^* back into the Lagrangian, obtaining the dual function,

$$D(\lambda) = \frac{1}{2} \|\theta^{(i-1)} + \lambda \delta - \theta^{(i-1)}\|^2 + \lambda(1 - (\theta^{(i-1)} + \lambda \delta) \cdot \delta) \quad [\text{B.17}]$$

$$= \frac{\lambda^2}{2} \|\delta\|^2 - \lambda^2 \|\delta\|^2 + \lambda(1 - \theta^{(i-1)} \cdot \delta) \quad [\text{B.18}]$$

$$= -\frac{\lambda^2}{2} \|\delta\|^2 + \lambda \ell^{(i)}(\theta^{(i-1)}). \quad [\text{B.19}]$$

Differentiating and solving for λ ,

$$\frac{\partial D}{\partial \lambda} = -\lambda \|\delta\|^2 + \ell^{(i)}(\theta^{(i-1)}) \quad [\text{B.20}]$$

$$\lambda^* = \frac{\ell^{(i)}(\theta^{(i-1)})}{\|\delta\|^2}. \quad [\text{B.21}]$$

The complete update equation is therefore:

$$\theta^* = \theta^{(i-1)} + \frac{\ell^{(i)}(\theta^{(i-1)})}{\|f(x^{(i)}, y^{(i)}) - f(x^{(i)}, \hat{y})\|^2} (f(x^{(i)}, y^{(i)}) - f(x^{(i)}, \hat{y})). \quad [\text{B.22}]$$

This update has strong intuitive support. The numerator of the learning rate grows with the loss. The denominator grows with the norm of the difference between the feature vectors associated with the correct and predicted label. If this norm is large, then the step with respect to each feature should be small, and vice versa.

10414

Bibliography

- 10415 Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis,
10416 J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia,
10417 R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore,
10418 D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A.
10419 Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Watten-
10420 berg, M. Wicke, Y. Yu, and X. Zheng (2016). Tensorflow: Large-scale machine learning
10421 on heterogeneous distributed systems. *CoRR abs/1603.04467*.
- 10422 Abend, O. and A. Rappoport (2017). The state of the art in semantic representation. In
10423 *Proceedings of the Association for Computational Linguistics (ACL)*.
- 10424 Abney, S., R. E. Schapire, and Y. Singer (1999). Boosting applied to tagging and PP attach-
10425 ment. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.
10426 132–134.
- 10427 Abney, S. P. (1987). *The English noun phrase in its sentential aspect*. Ph. D. thesis, Mas-
10428 sachusetts Institute of Technology.
- 10429 Abney, S. P. and M. Johnson (1991). Memory requirements and local ambiguities of pars-
10430 ing strategies. *Journal of Psycholinguistic Research* 20(3), 233–250.
- 10431 Adafre, S. F. and M. De Rijke (2006). Finding similar sentences across multiple languages
10432 in wikipedia. In *Proceedings of the Workshop on NEW TEXT Wikis and blogs and other*
10433 *dynamic text sources*.
- 10434 Ahn, D. (2006). The stages of event extraction. In *Proceedings of the Workshop on Annotating*
10435 *and Reasoning about Time and Events*, pp. 1–8. Association for Computational Linguistics.
- 10436 Aho, A. V., M. S. Lam, R. Sethi, and J. D. Ullman (2006). Compilers: Principles, techniques,
10437 & tools.
- 10438 Aikhenvald, A. Y. (2004). *Evidentiality*. Oxford University Press.

- 10439 Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on
10440 Automatic Control* 19(6), 716–723.
- 10441 Akmajian, A., R. A. Demers, A. K. Farmer, and R. M. Harnish (2010). *Linguistics: An
10442 introduction to language and communication* (Sixth ed.). Cambridge, MA: MIT press.
- 10443 Alfau, F. (1999). *Chromos*. Dalkey Archive Press.
- 10444 Allauzen, C., M. Riley, J. Schalkwyk, W. Skut, and M. Mohri (2007). OpenFst: A gen-
10445 eral and efficient weighted finite-state transducer library. In *International Conference on
10446 Implementation and Application of Automata*, pp. 11–23. Springer.
- 10447 Allen, J. F. (1984). Towards a general theory of action and time. *Artificial intelligence* 23(2),
10448 123–154.
- 10449 Allen, J. F., B. W. Miller, E. K. Ringger, and T. Sikorski (1996). A robust system for natural
10450 spoken dialogue. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
10451 62–70.
- 10452 Allen, J. F., L. K. Schubert, G. Ferguson, P. Heeman, C. H. Hwang, T. Kato, M. Light,
10453 N. Martin, B. Miller, M. Poesio, and D. Traum (1995). The TRAINS project: A case
10454 study in building a conversational planning agent. *Journal of Experimental & Theoretical
10455 Artificial Intelligence* 7(1), 7–48.
- 10456 Alm, C. O., D. Roth, and R. Sproat (2005). Emotions from text: machine learning for
10457 text-based emotion prediction. In *Proceedings of Empirical Methods for Natural Language
10458 Processing (EMNLP)*, pp. 579–586.
- 10459 Aluísio, S., J. Pelizzoni, A. Marchi, L. de Oliveira, R. Manenti, and V. Marquiafável (2003).
10460 An account of the challenge of tagging a reference corpus for Brazilian Portuguese.
10461 *Computational Processing of the Portuguese Language*, 194–194.
- 10462 Anand, P., M. Walker, R. Abbott, J. E. Fox Tree, R. Bowman, and M. Minor (2011). Cats rule
10463 and dogs drool!: Classifying stance in online debate. In *Proceedings of the 2nd Workshop
10464 on Computational Approaches to Subjectivity and Sentiment Analysis*, Portland, Oregon, pp.
10465 1–9. Association for Computational Linguistics.
- 10466 Anandkumar, A. and R. Ge (2016). Efficient approaches for escaping higher order saddle
10467 points in non-convex optimization. In *Proceedings of the Conference On Learning Theory
10468 (COLT)*, pp. 81–102.
- 10469 Anandkumar, A., R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky (2014). Tensor decompo-
10470 sitions for learning latent variable models. *The Journal of Machine Learning Research* 15(1),
10471 2773–2832.

- 10472 Ando, R. K. and T. Zhang (2005). A framework for learning predictive structures from
10473 multiple tasks and unlabeled data. *The Journal of Machine Learning Research* 6, 1817–
10474 1853.
- 10475 Andor, D., C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and
10476 M. Collins (2016). Globally normalized transition-based neural networks. In *Proceedings*
10477 of the Association for Computational Linguistics (ACL), pp. 2442–2452.
- 10478 Angeli, G., P. Liang, and D. Klein (2010). A simple domain-independent probabilistic ap-
10479 proach to generation. In *Proceedings of Empirical Methods for Natural Language Processing*
10480 (EMNLP), pp. 502–512.
- 10481 Antol, S., A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh
10482 (2015). Vqa: Visual question answering. In *Proceedings of the International Conference on*
10483 *Computer Vision (ICCV)*, pp. 2425–2433.
- 10484 Aronoff, M. (1976). *Word formation in generative grammar*. MIT Press.
- 10485 Arora, S. and B. Barak (2009). *Computational complexity: a modern approach*. Cambridge
10486 University Press.
- 10487 Arora, S., R. Ge, Y. Halpern, D. Mimmo, A. Moitra, D. Sontag, Y. Wu, and M. Zhu (2013).
10488 A practical algorithm for topic modeling with provable guarantees. In *Proceedings of the*
10489 *International Conference on Machine Learning (ICML)*, pp. 280–288.
- 10490 Arora, S., Y. Li, Y. Liang, T. Ma, and A. Risteski (2016). Linear algebraic structure of word
10491 senses, with applications to polysemy. *arXiv preprint arXiv:1601.03764*.
- 10492 Artstein, R. and M. Poesio (2008). Inter-coder agreement for computational linguistics.
10493 *Computational Linguistics* 34(4), 555–596.
- 10494 Artzi, Y. and L. Zettlemoyer (2013). Weakly supervised learning of semantic parsers for
10495 mapping instructions to actions. *Transactions of the Association for Computational Linguis-*
10496 *tics* 1, 49–62.
- 10497 Attardi, G. (2006). Experiments with a multilanguage non-projective dependency parser.
10498 In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 166–170.
- 10499 Auer, P. (2013). *Code-switching in conversation: Language, interaction and identity*. Routledge.
- 10500 Auer, S., C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives (2007). Dbpedia: A
10501 nucleus for a web of open data. *The semantic web*, 722–735.
- 10502 Austin, J. L. (1962). *How to do things with words*. Oxford University Press.

- 10503 Aw, A., M. Zhang, J. Xiao, and J. Su (2006). A phrase-based statistical model for SMS text
 10504 normalization. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
 10505 33–40.
- 10506 Ba, J. L., J. R. Kiros, and G. E. Hinton (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.
- 10508 Bagga, A. and B. Baldwin (1998a). Algorithms for scoring coreference chains. In *Proceedings of the Language Resources and Evaluation Conference*, pp. 563–566.
- 10510 Bagga, A. and B. Baldwin (1998b). Entity-based cross-document coreferencing using the
 10511 vector space model. In *Proceedings of the International Conference on Computational Lin-
 10512 guistics (COLING)*, pp. 79–85.
- 10513 Bahdanau, D., K. Cho, and Y. Bengio (2014). Neural machine translation by jointly learn-
 10514 ing to align and translate. In *Neural Information Processing Systems (NIPS)*.
- 10515 Baldwin, T. and S. N. Kim (2010). Multiword expressions. In *Handbook of natural language
 10516 processing*, Volume 2, pp. 267–292. Boca Raton, USA: CRC Press.
- 10517 Balle, B., A. Quattoni, and X. Carreras (2011). A spectral learning algorithm for finite state
 10518 transducers. In *Proceedings of the European Conference on Machine Learning and Principles
 10519 and Practice of Knowledge Discovery in Databases (ECML)*, pp. 156–171.
- 10520 Banarescu, L., C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight,
 10521 P. Koehn, M. Palmer, and N. Schneider (2013, August). Abstract meaning represen-
 10522 tation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability
 10523 with Discourse*, Sofia, Bulgaria, pp. 178–186. Association for Computational
 10524 Linguistics.
- 10525 Banko, M., M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni (2007). Open
 10526 information extraction from the web. In *Proceedings of the International Joint Conference
 10527 on Artificial Intelligence (IJCAI)*, pp. 2670–2676.
- 10528 Bansal, N., A. Blum, and S. Chawla (2004). Correlation clustering. *Machine Learning* 56(1–
 10529 3), 89–113.
- 10530 Barber, D. (2012). *Bayesian reasoning and machine learning*. Cambridge University Press.
- 10531 Barman, U., A. Das, J. Wagner, and J. Foster (2014, October). Code mixing: A challenge for
 10532 language identification in the language of social media. In *Proceedings of the First Work-
 10533 shop on Computational Approaches to Code Switching*, Doha, Qatar, pp. 13–23. Association
 10534 for Computational Linguistics.

- 10535 Barnickel, T., J. Weston, R. Collobert, H.-W. Mewes, and V. Stümpflen (2009). Large scale
10536 application of neural network based semantic role labeling for automated relation ex-
10537 traction from biomedical texts. *PLoS One* 4(7), e6393.
- 10538 Baron, A. and P. Rayson (2008). Vard2: A tool for dealing with spelling variation in his-
10539 torical corpora. In *Postgraduate conference in corpus linguistics*.
- 10540 Baroni, M., R. Bernardi, and R. Zamparelli (2014). Frege in space: A program for compo-
10541 sitional distributional semantics. *Linguistic Issues in Language Technologies*.
- 10542 Barzilay, R. and M. Lapata (2008, mar). Modeling local coherence: An Entity-Based ap-
10543 proach. *Computational Linguistics* 34(1), 1–34.
- 10544 Barzilay, R. and K. R. McKeown (2005). Sentence fusion for multidocument news summa-
10545 rization. *Computational Linguistics* 31(3), 297–328.
- 10546 Beesley, K. R. and L. Karttunen (2003). *Finite-state morphology*. Stanford, CA: Center for
10547 the Study of Language and Information.
- 10548 Bejan, C. A. and S. Harabagiu (2014). Unsupervised event coreference resolution. *Compu-*
10549 *tational Linguistics* 40(2), 311–347.
- 10550 Bell, E. T. (1934). Exponential numbers. *The American Mathematical Monthly* 41(7), 411–419.
- 10551 Bender, E. M. (2013, jun). *Linguistic Fundamentals for Natural Language Processing: 100*
10552 *Essentials from Morphology and Syntax*, Volume 6 of *Synthesis Lectures on Human Language*
10553 *Technologies*. Morgan & Claypool Publishers.
- 10554 Bengio, S., O. Vinyals, N. Jaitly, and N. Shazeer (2015). Scheduled sampling for sequence
10555 prediction with recurrent neural networks. In *Neural Information Processing Systems*
10556 (*NIPS*), pp. 1171–1179.
- 10557 Bengio, Y., R. Ducharme, P. Vincent, and C. Janvin (2003). A neural probabilistic language
10558 model. *The Journal of Machine Learning Research* 3, 1137–1155.
- 10559 Bengio, Y., P. Simard, and P. Frasconi (1994). Learning long-term dependencies with gra-
10560 dient descent is difficult. *IEEE Transactions on Neural Networks* 5(2), 157–166.
- 10561 Bengtsson, E. and D. Roth (2008). Understanding the value of features for coreference
10562 resolution. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
10563 pp. 294–303.
- 10564 Benjamini, Y. and Y. Hochberg (1995). Controlling the false discovery rate: a practical and
10565 powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B*
10566 (*Methodological*), 289–300.

- 10567 Berant, J., A. Chou, R. Frostig, and P. Liang (2013). Semantic parsing on freebase from
10568 question-answer pairs. In *Proceedings of Empirical Methods for Natural Language Processing* (EMNLP), pp. 1533–1544.
- 10570 Berant, J., V. Srikumar, P.-C. Chen, A. Vander Linden, B. Harding, B. Huang, P. Clark, and
10571 C. D. Manning (2014). Modeling biological processes for reading comprehension. In
10572 *Proceedings of Empirical Methods for Natural Language Processing* (EMNLP).
- 10573 Berg-Kirkpatrick, T., A. Bouchard-Côté, J. DeNero, and D. Klein (2010). Painless unsuper-
10574 vised learning with features. In *Proceedings of the North American Chapter of the Associa-*
10575 *tion for Computational Linguistics* (NAACL), pp. 582–590.
- 10576 Berg-Kirkpatrick, T., D. Burkett, and D. Klein (2012). An empirical investigation of sta-
10577 tistical significance in NLP. In *Proceedings of Empirical Methods for Natural Language*
10578 *Processing* (EMNLP), pp. 995–1005.
- 10579 Berger, A. L., V. J. D. Pietra, and S. A. D. Pietra (1996). A maximum entropy approach to
10580 natural language processing. *Computational linguistics* 22(1), 39–71.
- 10581 Bergsma, S., D. Lin, and R. Goebel (2008). Distributional identification of non-referential
10582 pronouns. In *Proceedings of the Association for Computational Linguistics* (ACL), pp. 10–18.
- 10583 Bernardi, R., R. Cakici, D. Elliott, A. Erdem, E. Erdem, N. Ikizler-Cinbis, F. Keller, A. Mus-
10584 cat, and B. Plank (2016). Automatic description generation from images: A survey of
10585 models, datasets, and evaluation measures. *Journal of Artificial Intelligence Research* 55,
10586 409–442.
- 10587 Bertsekas, D. P. (2012). Incremental gradient, subgradient, and proximal methods for
10588 convex optimization: A survey. See Sra et al. (2012).
- 10589 Bhatia, P., R. Guthrie, and J. Eisenstein (2016). Morphological priors for probabilistic neu-
10590 ral word embeddings. In *Proceedings of Empirical Methods for Natural Language Processing*
10591 (EMNLP).
- 10592 Bhatia, P., Y. Ji, and J. Eisenstein (2015). Better document-level sentiment analysis from
10593 first discourse parsing. In *Proceedings of Empirical Methods for Natural Language Processing*
10594 (EMNLP).
- 10595 Biber, D. (1991). *Variation across speech and writing*. Cambridge University Press.
- 10596 Bird, S., E. Klein, and E. Loper (2009). *Natural language processing with Python*. California:
10597 O'Reilly Media.
- 10598 Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

- 10599 Björkelund, A. and P. Nugues (2011). Exploring lexicalized features for coreference reso-
10600 lution. In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 45–50.
- 10601 Blackburn, P. and J. Bos (2005). *Representation and inference for natural language: A first*
10602 *course in computational semantics*. CSLI.
- 10603 Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM* 55(4), 77–84.
- 10604 Blei, D. M. (2014). Build, compute, critique, repeat: Data analysis with latent variable
10605 models. *Annual Review of Statistics and Its Application* 1, 203–232.
- 10606 Blei, D. M., A. Y. Ng, and M. I. Jordan (2003). Latent dirichlet allocation. *the Journal of*
10607 *machine Learning research* 3, 993–1022.
- 10608 Blitzer, J., M. Dredze, and F. Pereira (2007). Biographies, bollywood, boom-boxes and
10609 blenders: Domain adaptation for sentiment classification. In *Proceedings of the Associa-*
10610 *tion for Computational Linguistics (ACL)*, pp. 440–447.
- 10611 Blum, A. and T. Mitchell (1998). Combining labeled and unlabeled data with co-training.
10612 In *Proceedings of the Conference On Learning Theory (COLT)*, pp. 92–100.
- 10613 Bobrow, D. G., R. M. Kaplan, M. Kay, D. A. Norman, H. Thompson, and T. Winograd
10614 (1977). Gus, a frame-driven dialog system. *Artificial intelligence* 8(2), 155–173.
- 10615 Bochnet, B. (2010). Very high accuracy and fast dependency parsing is not a contradiction.
10616 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.
10617 89–97.
- 10618 Boitet, C. (1988). Pros and cons of the pivot and transfer approaches in multilingual ma-
10619 chine translation. *Readings in machine translation*, 273–279.
- 10620 Bojanowski, P., E. Grave, A. Joulin, and T. Mikolov (2017). Enriching word vectors with
10621 subword information. *Transactions of the Association for Computational Linguistics* 5, 135–
10622 146.
- 10623 Bollacker, K., C. Evans, P. Paritosh, T. Sturge, and J. Taylor (2008). Freebase: a collabora-
10624 tively created graph database for structuring human knowledge. In *Proceedings of the*
10625 *ACM International Conference on Management of Data (SIGMOD)*, pp. 1247–1250. AcM.
- 10626 Bolukbasi, T., K.-W. Chang, J. Y. Zou, V. Saligrama, and A. T. Kalai (2016). Man is to
10627 computer programmer as woman is to homemaker? debiasing word embeddings. In *Neural Information Processing Systems (NIPS)*, pp. 4349–4357.
- 10629 Bordes, A., N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko (2013). Translating
10630 embeddings for modeling multi-relational data. In *Neural Information Processing Systems*
10631 (*NIPS*), pp. 2787–2795.

- 10632 Bordes, A., J. Weston, R. Collobert, Y. Bengio, et al. (2011). Learning structured embed-
 10633 dings of knowledge bases. In *Proceedings of the National Conference on Artificial Intelligence
 10634 (AAAI)*, pp. 301–306.
- 10635 Borges, J. L. (1993). *Other Inquisitions 1937–1952*. University of Texas Press. Translated by
 10636 Ruth L. C. Simms.
- 10637 Botha, J. A. and P. Blunsom (2014). Compositional morphology for word representations
 10638 and language modelling. In *Proceedings of the International Conference on Machine Learn-
 10639 ing (ICML)*.
- 10640 Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*,
 10641 pp. 421–436. Springer.
- 10642 Bottou, L., F. E. Curtis, and J. Nocedal (2016). Optimization methods for large-scale ma-
 10643 chine learning. *arXiv preprint arXiv:1606.04838*.
- 10644 Bowman, S. R., L. Vilnis, O. Vinyals, A. Dai, R. Jozefowicz, and S. Bengio (2016). Gen-
 10645 erating sentences from a continuous space. In *Proceedings of the Conference on Natural
 10646 Language Learning (CoNLL)*, pp. 10–21.
- 10647 boyd, d. and K. Crawford (2012). Critical questions for big data. *Information, Communica-
 10648 tion & Society* 15(5), 662–679.
- 10649 Boyd, S. and L. Vandenberghe (2004). *Convex Optimization*. New York: Cambridge Uni-
 10650 versity Press.
- 10651 Branavan, S., H. Chen, J. Eisenstein, and R. Barzilay (2009). Learning document-level
 10652 semantic properties from free-text annotations. *Journal of Artificial Intelligence Re-
 10653 search* 34(2), 569–603.
- 10654 Branavan, S. R., H. Chen, L. S. Zettlemoyer, and R. Barzilay (2009). Reinforcement learning
 10655 for mapping instructions to actions. In *Proceedings of the Association for Computational
 10656 Linguistics (ACL)*, pp. 82–90.
- 10657 Braud, C., O. Lacroix, and A. Søgaard (2017). Does syntax help discourse segmenta-
 10658 tion? not so much. In *Proceedings of Empirical Methods for Natural Language Processing
 10659 (EMNLP)*, pp. 2432–2442.
- 10660 Briscoe, T. (2011). Introduction to formal semantics for natural language.
- 10661 Brown, P. F., J. Cocke, S. A. D. Pietra, V. J. D. Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer,
 10662 and P. S. Roossin (1990). A statistical approach to machine translation. *Computational
 10663 linguistics* 16(2), 79–85.

- 10664 Brown, P. F., P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai (1992). Class-based
10665 n-gram models of natural language. *Computational linguistics* 18(4), 467–479.
- 10666 Brown, P. F., V. J. D. Pietra, S. A. D. Pietra, and R. L. Mercer (1993). The mathematics
10667 of statistical machine translation: Parameter estimation. *Computational linguistics* 19(2),
10668 263–311.
- 10669 Brun, C. and C. Roux (2014). Décomposition des “hash tags” pour l’amélioration de la
10670 classification en polarité des “tweets”. *Proceedings of Traitement Automatique des Langues
10671 Naturelles*, 473–478.
- 10672 Bruni, E., N.-K. Tran, and M. Baroni (2014). Multimodal distributional semantics. *Journal
10673 of Artificial Intelligence Research* 49(2014), 1–47.
- 10674 Bullinaria, J. A. and J. P. Levy (2007). Extracting semantic representations from word co-
10675 occurrence statistics: A computational study. *Behavior research methods* 39(3), 510–526.
- 10676 Bunescu, R. C. and R. J. Mooney (2005). A shortest path dependency kernel for relation
10677 extraction. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
10678 pp. 724–731.
- 10679 Bunescu, R. C. and M. Pasca (2006). Using encyclopedic knowledge for named entity
10680 disambiguation. In *Proceedings of the European Chapter of the Association for Computational
10681 Linguistics (EACL)*, pp. 9–16.
- 10682 Burstein, J., D. Marcu, and K. Knight (2003). Finding the WRITE stuff: Automatic identi-
10683 fication of discourse structure in student essays. *IEEE Intelligent Systems* 18(1), 32–39.
- 10684 Burstein, J., J. Tetreault, and S. Andreyev (2010). Using entity-based features to model
10685 coherence in student essays. In *Human language technologies: The 2010 annual conference
10686 of the North American chapter of the Association for Computational Linguistics*, pp. 681–684.
10687 Association for Computational Linguistics.
- 10688 Burstein, J., J. Tetreault, and M. Chodorow (2013). Holistic discourse coherence annotation
10689 for noisy essay writing. *Dialogue & Discourse* 4(2), 34–52.
- 10690 Cai, Q. and A. Yates (2013). Large-scale semantic parsing via schema matching and lexicon
10691 extension. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 423–
10692 433.
- 10693 Caliskan, A., J. J. Bryson, and A. Narayanan (2017). Semantics derived automatically from
10694 language corpora contain human-like biases. *Science* 356(6334), 183–186.
- 10695 Canny, J. (1987). A computational approach to edge detection. In *Readings in Computer
10696 Vision*, pp. 184–203. Elsevier.

- 10697 Cappé, O. and E. Moulines (2009). On-line expectation–maximization algorithm for latent
10698 data models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 71(3),
10699 593–613.
- 10700 Carbonell, J. and J. Goldstein (1998). The use of mmr, diversity-based reranking for re-
10701 ordering documents and producing summaries. In *Proceedings of ACM SIGIR conference*
10702 on Research and development in information retrieval, pp. 335–336.
- 10703 Carbonell, J. R. (1970). Mixed-initiative man-computer instructional dialogues. Technical
10704 report, BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS.
- 10705 Cardie, C. and K. Wagstaff (1999). Noun phrase coreference as clustering. In *Proceedings*
10706 of *Empirical Methods for Natural Language Processing (EMNLP)*, pp. 82–89.
- 10707 Carletta, J. (1996). Assessing agreement on classification tasks: the kappa statistic. *Com-
10708 putational linguistics* 22(2), 249–254.
- 10709 Carletta, J. (2007). Unleashing the killer corpus: experiences in creating the multi-
10710 everything ami meeting corpus. *Language Resources and Evaluation* 41(2), 181–190.
- 10711 Carlson, L. and D. Marcu (2001). Discourse tagging reference manual. Technical Report
10712 ISI-TR-545, Information Sciences Institute.
- 10713 Carlson, L., M. E. Okurowski, and D. Marcu (2002). RST discourse treebank. Linguistic
10714 Data Consortium, University of Pennsylvania.
- 10715 Carpenter, B. (1997). *Type-logical semantics*. Cambridge, MA: MIT Press.
- 10716 Carreras, X., M. Collins, and T. Koo (2008). Tag, dynamic programming, and the percep-
10717 tron for efficient, feature-rich parsing. In *Proceedings of the Conference on Natural Language*
10718 *Learning (CoNLL)*, pp. 9–16.
- 10719 Carreras, X. and L. Màrquez (2005). Introduction to the conll-2005 shared task: Semantic
10720 role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language*
10721 *Learning*, pp. 152–164. Association for Computational Linguistics.
- 10722 Carroll, L. (1917). *Through the looking glass: And what Alice found there*. Chicago: Rand,
10723 McNally.
- 10724 Chambers, N. and D. Jurafsky (2008). Jointly combining implicit constraints improves
10725 temporal ordering. In *Proceedings of Empirical Methods for Natural Language Processing*
10726 (*EMNLP*), pp. 698–706.
- 10727 Chang, K.-W., A. Krishnamurthy, A. Agarwal, H. Daume III, and J. Langford (2015).
10728 Learning to search better than your teacher. In *Proceedings of the International Confer-
10729 ence on Machine Learning (ICML)*.

- 10730 Chang, M.-W., L. Ratinov, and D. Roth (2007). Guiding semi-supervision with constraint-
10731 driven learning. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
10732 280–287.
- 10733 Chang, M.-W., L.-A. Ratinov, N. Rizzolo, and D. Roth (2008). Learning and inference with
10734 constraints. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp.
10735 1513–1518.
- 10736 Chapman, W. W., W. Bridewell, P. Hanbury, G. F. Cooper, and B. G. Buchanan (2001). A
10737 simple algorithm for identifying negated findings and diseases in discharge summaries.
10738 *Journal of biomedical informatics* 34(5), 301–310.
- 10739 Charniak, E. (1997). Statistical techniques for natural language parsing. *AI magazine* 18(4),
10740 33–43.
- 10741 Charniak, E. and M. Johnson (2005). Coarse-to-fine n-best parsing and maxent discrimi-
10742 native reranking. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
10743 173–180.
- 10744 Chelba, C. and A. Acero (2006). Adaptation of maximum entropy capitalizer: Little data
10745 can help a lot. *Computer Speech & Language* 20(4), 382–399.
- 10746 Chelba, C., T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson (2013).
10747 One billion word benchmark for measuring progress in statistical language modeling.
10748 *arXiv preprint arXiv:1312.3005*.
- 10749 Chen, D., J. Bolton, and C. D. Manning (2016). A thorough examination of the CNN/Daily
10750 Mail reading comprehension task. In *Proceedings of the Association for Computational
10751 Linguistics (ACL)*.
- 10752 Chen, D. and C. D. Manning (2014). A fast and accurate dependency parser using neural
10753 networks. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
10754 pp. 740–750.
- 10755 Chen, D. L. and R. J. Mooney (2008). Learning to sportscast: a test of grounded language
10756 acquisition. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp.
10757 128–135.
- 10758 Chen, H., S. Branavan, R. Barzilay, and D. R. Karger (2009). Content modeling using latent
10759 permutations. *Journal of Artificial Intelligence Research* 36(1), 129–163.
- 10760 Chen, M., Z. Xu, K. Weinberger, and F. Sha (2012). Marginalized denoising autoencoders
10761 for domain adaptation. In *Proceedings of the International Conference on Machine Learning
10762 (ICML)*.

- 10763 Chen, M. X., O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones, N. Parmar,
 10764 M. Schuster, Z. Chen, Y. Wu, and M. Hughes (2018). The best of both worlds: Combin-
 10765 ing recent advances in neural machine translation. In *Proceedings of the Association for*
 10766 *Computational Linguistics (ACL)*.
- 10767 Chen, S. F. and J. Goodman (1999). An empirical study of smoothing techniques for lan-
 10768 guage modeling. *Computer Speech & Language* 13(4), 359–393.
- 10769 Chen, T. and C. Guestrin (2016). Xgboost: A scalable tree boosting system. In *Proceedings*
 10770 *of Knowledge Discovery and Data Mining (KDD)*, pp. 785–794.
- 10771 Chen, X., X. Qiu, C. Zhu, P. Liu, and X. Huang (2015). Long short-term memory neural
 10772 networks for chinese word segmentation. In *Proceedings of Empirical Methods for Natural*
 10773 *Language Processing (EMNLP)*, pp. 1197–1206.
- 10774 Chen, Y., S. Gilroy, A. Malletti, K. Knight, and J. May (2018). Recurrent neural networks
 10775 as weighted language recognizers. In *Proceedings of the North American Chapter of the*
 10776 *Association for Computational Linguistics (NAACL)*.
- 10777 Chen, Z. and H. Ji (2009). Graph-based event coreference resolution. In *Proceedings of*
 10778 *the 2009 Workshop on Graph-based Methods for Natural Language Processing*, pp. 54–57.
 10779 Association for Computational Linguistics.
- 10780 Cheng, X. and D. Roth (2013). Relational inference for wikification. In *Proceedings of*
 10781 *Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1787–1796.
- 10782 Chiang, D. (2007). Hierarchical phrase-based translation. *Computational Linguistics* 33(2),
 10783 201–228.
- 10784 Chiang, D., J. Graehl, K. Knight, A. Pauls, and S. Ravi (2010). Bayesian inference for
 10785 finite-state transducers. In *Proceedings of the North American Chapter of the Association for*
 10786 *Computational Linguistics (NAACL)*, pp. 447–455.
- 10787 Cho, K. (2015). Natural language understanding with distributed representation.
 10788 *CoRR abs/1511.07916*.
- 10789 Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and
 10790 Y. Bengio (2014). Learning phrase representations using rnn encoder-decoder for sta-
 10791 tistical machine translation. In *Proceedings of Empirical Methods for Natural Language*
 10792 *Processing (EMNLP)*.
- 10793 Chomsky, N. (1957). *Syntactic structures*. The Hague: Mouton & Co.
- 10794 Chomsky, N. (1982). *Some concepts and consequences of the theory of government and binding*,
 10795 Volume 6. MIT press.

- 10796 Choromanska, A., M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun (2015). The loss
10797 surfaces of multilayer networks. In *Proceedings of Artificial Intelligence and Statistics (AIS-*
10798 *TATS)*, pp. 192–204.
- 10799 Christensen, J., S. Soderland, O. Etzioni, et al. (2010). Semantic role labeling for open
10800 information extraction. In *Proceedings of the Workshop on Formalisms and Methodology for*
10801 *Learning by Reading*, pp. 52–60. Association for Computational Linguistics.
- 10802 Christodoulopoulos, C., S. Goldwater, and M. Steedman (2010). Two decades of unsuper-
10803 vised pos induction: How far have we come? In *Proceedings of Empirical Methods for*
10804 *Natural Language Processing (EMNLP)*, pp. 575–584.
- 10805 Chu, Y.-J. and T.-H. Liu (1965). On shortest arborescence of a directed graph. *Scientia*
10806 *Sinica* 14(10), 1396–1400.
- 10807 Chung, C. and J. W. Pennebaker (2007). The psychological functions of function words.
10808 In K. Fiedler (Ed.), *Social communication*, pp. 343–359. New York and Hove: Psychology
10809 Press.
- 10810 Church, K. (2011). A pendulum swung too far. *Linguistic Issues in Language Technology* 6(5),
10811 1–27.
- 10812 Church, K. W. (2000). Empirical estimates of adaptation: the chance of two Noriega-
10813 s is closer to $p/2$ than p^2 . In *Proceedings of the International Conference on Computational*
10814 *Linguistics (COLING)*, pp. 180–186.
- 10815 Church, K. W. and P. Hanks (1990). Word association norms, mutual information, and
10816 lexicography. *Computational linguistics* 16(1), 22–29.
- 10817 Ciaramita, M. and M. Johnson (2003). Supersense tagging of unknown nouns in wordnet.
10818 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 168–
10819 175.
- 10820 Clark, K. and C. D. Manning (2015). Entity-centric coreference resolution with model
10821 stacking. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1405–
10822 1415.
- 10823 Clark, K. and C. D. Manning (2016). Improving coreference resolution by learning entity-
10824 level distributed representations. In *Proceedings of the Association for Computational Lin-*
10825 *guistics (ACL)*.
- 10826 Clark, P. (2015). Elementary school science and math tests as a driver for ai: take the aristo
10827 challenge! In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp.
10828 4019–4021.

- 10829 Clarke, J., D. Goldwasser, M.-W. Chang, and D. Roth (2010). Driving semantic parsing
10830 from the world's response. In *Proceedings of the Conference on Natural Language Learning*
10831 (*CoNLL*), pp. 18–27.
- 10832 Clarke, J. and M. Lapata (2008). Global inference for sentence compression: An integer
10833 linear programming approach. *Journal of Artificial Intelligence Research* 31, 399–429.
- 10834 Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and psychologi-*
10835 *cal measurement* 20(1), 37–46.
- 10836 Cohen, S. (2016). *Bayesian analysis in natural language processing*. Synthesis Lectures on
10837 Human Language Technologies. San Rafael, CA: Morgan & Claypool Publishers.
- 10838 Collier, N., C. Nobata, and J.-i. Tsujii (2000). Extracting the names of genes and gene
10839 products with a hidden markov model. In *Proceedings of the International Conference on*
10840 *Computational Linguistics (COLING)*, pp. 201–207.
- 10841 Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *Proceed-*
10842 *ings of the Association for Computational Linguistics (ACL)*, pp. 16–23.
- 10843 Collins, M. (2002). Discriminative training methods for hidden markov models: theory
10844 and experiments with perceptron algorithms. In *Proceedings of Empirical Methods for*
10845 *Natural Language Processing (EMNLP)*, pp. 1–8.
- 10846 Collins, M. (2013). Notes on natural language processing. <http://www.cs.columbia.edu/~mcollins/notes-spring2013.html>.
- 10847
- 10848 Collins, M. and T. Koo (2005). Discriminative reranking for natural language parsing.
10849 *Computational Linguistics* 31(1), 25–70.
- 10850 Collins, M. and B. Roark (2004). Incremental parsing with the perceptron algorithm. In
10851 *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pp.
10852 111. Association for Computational Linguistics.
- 10853 Collobert, R., K. Kavukcuoglu, and C. Farabet (2011). Torch7: A matlab-like environment
10854 for machine learning. Technical Report EPFL-CONF-192376, EPFL.
- 10855 Collobert, R. and J. Weston (2008). A unified architecture for natural language process-
10856 ing: Deep neural networks with multitask learning. In *Proceedings of the International*
10857 *Conference on Machine Learning (ICML)*, pp. 160–167.
- 10858 Collobert, R., J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa (2011). Nat-
10859 ural language processing (almost) from scratch. *Journal of Machine Learning Research* 12,
10860 2493–2537.

- 10861 Colton, S., J. Goodwin, and T. Veale (2012). Full-face poetry generation. In *Proceedings of*
10862 *the International Conference on Computational Creativity*, pp. 95–102.
- 10863 Conneau, A., D. Kiela, H. Schwenk, L. Barrault, and A. Bordes (2017). Supervised learning
10864 of universal sentence representations from natural language inference data. In *Proceed-*
10865 *ings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 681–691.
- 10866 Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2009). *Introduction to algorithms*
10867 (third ed.). MIT press.
- 10868 Cotterell, R., H. Schütze, and J. Eisner (2016). Morphological smoothing and extrapolation
10869 of word embeddings. In *Proceedings of the Association for Computational Linguistics (ACL)*,
10870 pp. 1651–1660.
- 10871 Coviello, L., Y. Sohn, A. D. Kramer, C. Marlow, M. Franceschetti, N. A. Christakis, and
10872 J. H. Fowler (2014). Detecting emotional contagion in massive social networks. *PloS*
10873 *one* 9(3), e90315.
- 10874 Covington, M. A. (2001). A fundamental algorithm for dependency parsing. In *Proceedings*
10875 *of the 39th annual ACM southeast conference*, pp. 95–102.
- 10876 Crammer, K., O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer (2006, December).
10877 Online passive-aggressive algorithms. *The Journal of Machine Learning Research* 7, 551–
10878 585.
- 10879 Crammer, K. and Y. Singer (2001). Pranking with ranking. In *Neural Information Processing*
10880 *Systems (NIPS)*, pp. 641–647.
- 10881 Creutz, M. and K. Lagus (2007). Unsupervised models for morpheme segmentation and
10882 morphology learning. *ACM Transactions on Speech and Language Processing (TSLP)* 4(1),
10883 3.
- 10884 Cross, J. and L. Huang (2016). Span-based constituency parsing with a structure-label
10885 system and provably optimal dynamic oracles. In *Proceedings of Empirical Methods for*
10886 *Natural Language Processing (EMNLP)*, pp. 1–11.
- 10887 Cucerzan, S. (2007). Large-scale named entity disambiguation based on wikipedia data.
10888 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 10889 Cui, H., R. Sun, K. Li, M.-Y. Kan, and T.-S. Chua (2005). Question answering passage
10890 retrieval using dependency relations. In *Proceedings of the 28th annual international ACM*
10891 *SIGIR conference on Research and development in information retrieval*, pp. 400–407. ACM.
- 10892 Cui, Y., Z. Chen, S. Wei, S. Wang, T. Liu, and G. Hu (2017). Attention-over-attention neural
10893 networks for reading comprehension. In *Proceedings of the Association for Computational*
10894 *Linguistics (ACL)*.

- 10895 Culotta, A. and J. Sorensen (2004). Dependency tree kernels for relation extraction. In
 10896 *Proceedings of the Association for Computational Linguistics (ACL)*.
- 10897 Culotta, A., M. Wick, and A. McCallum (2007). First-order probabilistic models for coref-
 10898 erence resolution. In *Proceedings of the North American Chapter of the Association for Com-*
 10899 *putational Linguistics (NAACL)*, pp. 81–88.
- 10900 Curry, H. B. and R. Feys (1958). *Combinatory Logic*, Volume I. Amsterdam: North Holland.
- 10901 Danescu-Niculescu-Mizil, C., M. Sudhof, D. Jurafsky, J. Leskovec, and C. Potts (2013). A
 10902 computational approach to politeness with application to social factors. In *Proceedings*
 10903 *of the Association for Computational Linguistics (ACL)*, pp. 250–259.
- 10904 Das, D., D. Chen, A. F. Martins, N. Schneider, and N. A. Smith (2014). Frame-semantic
 10905 parsing. *Computational Linguistics* 40(1), 9–56.
- 10906 Daumé III, H. (2007). Frustratingly easy domain adaptation. In *Proceedings of the Associa-*
 10907 *tion for Computational Linguistics (ACL)*.
- 10908 Daumé III, H., J. Langford, and D. Marcu (2009). Search-based structured prediction.
 10909 *Machine learning* 75(3), 297–325.
- 10910 Daumé III, H. and D. Marcu (2005). A large-scale exploration of effective global features
 10911 for a joint entity detection and tracking model. In *Proceedings of Empirical Methods for*
 10912 *Natural Language Processing (EMNLP)*, pp. 97–104.
- 10913 Dauphin, Y. N., R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio (2014). Identi-
 10914 fying and attacking the saddle point problem in high-dimensional non-convex opti-
 10915 mization. In *Neural Information Processing Systems (NIPS)*, pp. 2933–2941.
- 10916 Davidson, D. (1967). The logical form of action sentences. In N. Rescher (Ed.), *The Logic of*
 10917 *Decision and Action*. Pittsburgh: University of Pittsburgh Press.
- 10918 De Gispert, A. and J. B. Marino (2006). Catalan-english statistical machine translation
 10919 without parallel corpus: bridging through spanish. In *Proc. of 5th International Conference*
 10920 *on Language Resources and Evaluation (LREC)*, pp. 65–68. Citeseer.
- 10921 De Marneffe, M.-C. and C. D. Manning (2008). The stanford typed dependencies represen-
 10922 tation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain*
 10923 *Parser Evaluation*, pp. 1–8. Association for Computational Linguistics.
- 10924 Dean, J. and S. Ghemawat (2008). Mapreduce: simplified data processing on large clusters.
 10925 *Communications of the ACM* 51(1), 107–113.
- 10926 Deerwester, S. C., S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman (1990).
 10927 Indexing by latent semantic analysis. *JASIS* 41(6), 391–407.

- 10928 Dehdari, J. (2014). *A Neurophysiologically-Inspired Statistical Language Model*. Ph. D. thesis,
10929 The Ohio State University.
- 10930 Deisenroth, M. P., A. A. Faisal, and C. S. Ong (2018). *Mathematics For Machine Learning*.
10931 Cambridge UP.
- 10932 Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). Maximum likelihood from incom-
10933 plete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Method-
10934 ological)*, 1–38.
- 10935 Denis, P. and J. Baldridge (2007). A ranking approach to pronoun resolution. In *IJCAI*.
- 10936 Denis, P. and J. Baldridge (2008). Specialized models and ranking for coreference resolu-
10937 tion. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*,
10938 EMNLP '08, Stroudsburg, PA, USA, pp. 660–669. Association for Computational Lin-
10939 guistics.
- 10940 Denis, P. and J. Baldridge (2009). Global joint models for coreference resolution and named
10941 entity classification. *Procesamiento del Lenguaje Natural* 42.
- 10942 Derrida, J. (1985). Des tours de babel. In J. Graham (Ed.), *Difference in translation*. Ithaca,
10943 NY: Cornell University Press.
- 10944 Dhingra, B., H. Liu, Z. Yang, W. W. Cohen, and R. Salakhutdinov (2017). Gated-attention
10945 readers for text comprehension. In *Proceedings of the Association for Computational Lin-
10946 guistics (ACL)*.
- 10947 Diaconis, P. and B. Skyrms (2017). *Ten Great Ideas About Chance*. Princeton University
10948 Press.
- 10949 Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classifica-
10950 tion learning algorithms. *Neural computation* 10(7), 1895–1923.
- 10951 Dietterich, T. G., R. H. Lathrop, and T. Lozano-Pérez (1997). Solving the multiple instance
10952 problem with axis-parallel rectangles. *Artificial intelligence* 89(1), 31–71.
- 10953 Dimitrova, L., N. Ide, V. Petkevic, T. Erjavec, H. J. Kaalep, and D. Tufis (1998). Multext-
10954 east: Parallel and comparable corpora and lexicons for six central and eastern european
10955 languages. In *Proceedings of the 17th international conference on Computational linguistics-
10956 Volume 1*, pp. 315–319. Association for Computational Linguistics.
- 10957 Doddington, G. R., A. Mitchell, M. A. Przybocki, L. A. Ramshaw, S. Strassel, and R. M.
10958 Weischedel (2004). The automatic content extraction (ace) program-tasks, data, and
10959 evaluation. In *Proceedings of the Language Resources and Evaluation Conference*, pp. 837–
10960 840.

- 10961 dos Santos, C., B. Xiang, and B. Zhou (2015). Classifying relations by ranking with con-
 10962 volutional neural networks. In *Proceedings of the Association for Computational Linguistics*
 10963 (ACL), pp. 626–634.
- 10964 Dowty, D. (1991). Thematic proto-roles and argument selection. *Language*, 547–619.
- 10965 Dredze, M., P. McNamee, D. Rao, A. Gerber, and T. Finin (2010). Entity disambiguation
 10966 for knowledge base population. In *Proceedings of the 23rd International Conference on*
 10967 *Computational Linguistics*, pp. 277–285. Association for Computational Linguistics.
- 10968 Dredze, M., M. J. Paul, S. Bergsma, and H. Tran (2013). Carmen: A Twitter geolocation
 10969 system with applications to public health. In *AAAI workshop on expanding the boundaries*
 10970 *of health informatics using AI (HIAI)*, pp. 20–24.
- 10971 Dreyfus, H. L. (1992). *What computers still can't do: a critique of artificial reason*. MIT press.
- 10972 Du, L., W. Buntine, and M. Johnson (2013). Topic segmentation with a structured topic
 10973 model. In *Proceedings of the North American Chapter of the Association for Computational*
 10974 *Linguistics* (NAACL), pp. 190–200.
- 10975 Duchi, J., E. Hazan, and Y. Singer (2011). Adaptive subgradient methods for online learn-
 10976 ing and stochastic optimization. *The Journal of Machine Learning Research* 12, 2121–2159.
- 10977 Dunietz, J., L. Levin, and J. Carbonell (2017). The because corpus 2.0: Annotating causality
 10978 and overlapping relations. In *Proceedings of the Linguistic Annotation Workshop*.
- 10979 Durrett, G., T. Berg-Kirkpatrick, and D. Klein (2016). Learning-based single-document
 10980 summarization with compression and anaphoricity constraints. In *Proceedings of the*
 10981 *Association for Computational Linguistics* (ACL), pp. 1998–2008.
- 10982 Durrett, G. and D. Klein (2013). Easy victories and uphill battles in coreference resolution.
 10983 In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.
- 10984 Durrett, G. and D. Klein (2015). Neural crf parsing. In *Proceedings of the Association for*
 10985 *Computational Linguistics* (ACL).
- 10986 Dyer, C., M. Ballesteros, W. Ling, A. Matthews, and N. A. Smith (2015). Transition-based
 10987 dependency parsing with stack long short-term memory. In *Proceedings of the Association*
 10988 *for Computational Linguistics* (ACL), pp. 334–343.
- 10989 Dyer, C., A. Kuncoro, M. Ballesteros, and N. A. Smith (2016). Recurrent neural network
 10990 grammars. In *Proceedings of the North American Chapter of the Association for Computational*
 10991 *Linguistics* (NAACL), pp. 199–209.
- 10992 Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of*
 10993 *Standards B* 71(4), 233–240.

- 10994 Efron, B. and R. J. Tibshirani (1993). An introduction to the bootstrap: Monographs on
10995 statistics and applied probability, vol. 57. *New York and London: Chapman and Hall/CRC*.
- 10996 Eisenstein, J. (2009). Hierarchical text segmentation from multi-scale lexical cohesion. In
10997 *Proceedings of the North American Chapter of the Association for Computational Linguistics*
10998 (NAACL).
- 10999 Eisenstein, J. and R. Barzilay (2008). Bayesian unsupervised topic segmentation. In *Pro-*
11000 *ceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11001 Eisner, J. (1997). State-of-the-art algorithms for minimum spanning trees: A tutorial dis-
11002 cussion.
- 11003 Eisner, J. (2000). Bilexical grammars and their cubic-time parsing algorithms. In *Advances*
11004 *in probabilistic and other parsing technologies*, pp. 29–61. Springer.
- 11005 Eisner, J. (2002). Parameter estimation for probabilistic finite-state transducers. In *Proced-*
11006 *ings of the Association for Computational Linguistics (ACL)*, pp. 1–8.
- 11007 Eisner, J. (2016). Inside-outside and forward-backward algorithms are just backprop. In
11008 *Proceedings of the Workshop on Structured Prediction for NLP*, pp. 1–17.
- 11009 Eisner, J. M. (1996). Three new probabilistic models for dependency parsing: An explo-
11010 ration. In *Proceedings of the International Conference on Computational Linguistics (COL-*
11011 *ING)*, pp. 340–345.
- 11012 Ekman, P. (1992). Are there basic emotions? *Psychological Review* 99(3), 550–553.
- 11013 Elman, J. L. (1990). Finding structure in time. *Cognitive science* 14(2), 179–211.
- 11014 Elman, J. L., E. A. Bates, M. H. Johnson, A. Karmiloff-Smith, D. Parisi, and K. Plunkett
11015 (1998). *Rethinking innateness: A connectionist perspective on development*, Volume 10. MIT
11016 press.
- 11017 Elsner, M. and E. Charniak (2010). Disentangling chat. *Computational Linguistics* 36(3),
11018 389–409.
- 11019 Esuli, A. and F. Sebastiani (2006). Sentiwordnet: A publicly available lexical resource for
11020 opinion mining. In *LREC*, Volume 6, pp. 417–422. Citeseer.
- 11021 Etzioni, O., A. Fader, J. Christensen, S. Soderland, and M. Mausam (2011). Open informa-
11022 tion extraction: The second generation. In *Proceedings of the International Joint Conference*
11023 *on Artificial Intelligence (IJCAI)*, pp. 3–10.

- 11024 Faruqui, M., J. Dodge, S. K. Jauhar, C. Dyer, E. Hovy, and N. A. Smith (2015). Retrofitting
11025 word vectors to semantic lexicons. In *Proceedings of the North American Chapter of the*
11026 *Association for Computational Linguistics (NAACL)*.
- 11027 Faruqui, M. and C. Dyer (2014). Improving vector space word representations using mul-
11028 tilingual correlation. In *Proceedings of the European Chapter of the Association for Compu-*
11029 *tational Linguistics (EACL)*, pp. 462–471.
- 11030 Faruqui, M., R. McDonald, and R. Soricut (2016). Morpho-syntactic lexicon generation
11031 using graph-based semi-supervised learning. *Transactions of the Association for Compu-*
11032 *tational Linguistics* 4, 1–16.
- 11033 Faruqui, M., Y. Tsvetkov, P. Rastogi, and C. Dyer (2016, August). Problems with evaluation
11034 of word embeddings using word similarity tasks. In *Proceedings of the 1st Workshop on*
11035 *Evaluating Vector-Space Representations for NLP*, Berlin, Germany, pp. 30–35. Association
11036 for Computational Linguistics.
- 11037 Fellbaum, C. (2010). *WordNet*. Springer.
- 11038 Feng, V. W., Z. Lin, and G. Hirst (2014). The impact of deep hierarchical discourse struc-
11039 tures in the evaluation of text coherence. In *Proceedings of the International Conference on*
11040 *Computational Linguistics (COLING)*, pp. 940–949.
- 11041 Feng, X., L. Huang, D. Tang, H. Ji, B. Qin, and T. Liu (2016). A language-independent
11042 neural network for event detection. In *Proceedings of the Association for Computational*
11043 *Linguistics (ACL)*, pp. 66–71.
- 11044 Fernandes, E. R., C. N. dos Santos, and R. L. Milidiú (2014). Latent trees for coreference
11045 resolution. *Computational Linguistics*.
- 11046 Ferrucci, D., E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W.
11047 Murdock, E. Nyberg, J. Prager, et al. (2010). Building Watson: An overview of the
11048 DeepQA project. *AI magazine* 31(3), 59–79.
- 11049 Ficler, J. and Y. Goldberg (2017, September). Controlling linguistic style aspects in neural
11050 language generation. In *Proceedings of the Workshop on Stylistic Variation*, Copenhagen,
11051 Denmark, pp. 94–104. Association for Computational Linguistics.
- 11052 Filippova, K. and M. Strube (2008). Sentence fusion via dependency graph compression.
11053 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 177–
11054 185.
- 11055 Fillmore, C. J. (1968). The case for case. In E. Bach and R. Harms (Eds.), *Universals in*
11056 *linguistic theory*. Holt, Rinehart, and Winston.

- 11057 Fillmore, C. J. (1976). Frame semantics and the nature of language. *Annals of the New York
11058 Academy of Sciences* 280(1), 20–32.
- 11059 Fillmore, C. J. and C. Baker (2009). A frames approach to semantic analysis. In *The Oxford
11060 Handbook of Linguistic Analysis*. Oxford University Press.
- 11061 Finkel, J. R., T. Grenager, and C. Manning (2005). Incorporating non-local information
11062 into information extraction systems by gibbs sampling. In *Proceedings of the Association
11063 for Computational Linguistics (ACL)*, pp. 363–370.
- 11064 Finkel, J. R., T. Grenager, and C. D. Manning (2007). The infinite tree. In *Proceedings of the
11065 Association for Computational Linguistics (ACL)*, pp. 272–279.
- 11066 Finkel, J. R., A. Kleeman, and C. D. Manning (2008). Efficient, feature-based, conditional
11067 random field parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*,
11068 pp. 959–967.
- 11069 Finkel, J. R. and C. Manning (2009). Hierarchical bayesian domain adaptation. In *Proceed-
11070 ings of the North American Chapter of the Association for Computational Linguistics (NAACL)*,
11071 pp. 602–610.
- 11072 Finkel, J. R. and C. D. Manning (2008). Enforcing transitivity in coreference resolution.
11073 In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics
11074 on Human Language Technologies: Short Papers*, pp. 45–48. Association for Computational
11075 Linguistics.
- 11076 Finkelstein, L., E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin
11077 (2002). Placing search in context: The concept revisited. *ACM Transactions on Information
11078 Systems* 20(1), 116–131.
- 11079 Firth, J. R. (1957). *Papers in Linguistics 1934-1951*. Oxford University Press.
- 11080 Flanigan, J., S. Thomson, J. Carbonell, C. Dyer, and N. A. Smith (2014). A discrimina-
11081 tive graph-based parser for the abstract meaning representation. In *Proceedings of the
11082 Association for Computational Linguistics (ACL)*, pp. 1426–1436.
- 11083 Foltz, P. W., W. Kintsch, and T. K. Landauer (1998). The measurement of textual coherence
11084 with latent semantic analysis. *Discourse processes* 25(2-3), 285–307.
- 11085 Fordyce, C. S. (2007). Overview of the iwslt 2007 evaluation campaign. In *International
11086 Workshop on Spoken Language Translation (IWSLT) 2007*.
- 11087 Fox, H. (2002). Phrasal cohesion and statistical machine translation. In *Proceedings of
11088 Empirical Methods for Natural Language Processing (EMNLP)*, pp. 304–3111.

- 11089 Francis, W. and H. Kucera (1982). *Frequency analysis of English usage*. Houghton Mifflin
11090 Company.
- 11091 Francis, W. N. (1964). A standard sample of present-day English for use with digital
11092 computers. Report to the U.S Office of Education on Cooperative Research Project No.
11093 E-007.
- 11094 Freund, Y. and R. E. Schapire (1999). Large margin classification using the perceptron
11095 algorithm. *Machine learning* 37(3), 277–296.
- 11096 Fromkin, V., R. Rodman, and N. Hyams (2013). *An introduction to language*. Cengage
11097 Learning.
- 11098 Fundel, K., R. Küffner, and R. Zimmer (2007). Relex – relation extraction using depen-
11099 dency parse trees. *Bioinformatics* 23(3), 365–371.
- 11100 Gabow, H. N., Z. Galil, T. Spencer, and R. E. Tarjan (1986). Efficient algorithms for finding
11101 minimum spanning trees in undirected and directed graphs. *Combinatorica* 6(2), 109–
11102 122.
- 11103 Gabrilovich, E. and S. Markovitch (2007). Computing semantic relatedness using
11104 wikipedia-based explicit semantic analysis. In *Proceedings of the International Joint Con-*
11105 *ference on Artificial Intelligence (IJCAI)*, Volume 7, pp. 1606–1611.
- 11106 Gage, P. (1994). A new algorithm for data compression. *The C Users Journal* 12(2), 23–38.
- 11107 Gale, W. A., K. W. Church, and D. Yarowsky (1992). One sense per discourse. In *Pro-*
11108 *ceedings of the workshop on Speech and Natural Language*, pp. 233–237. Association for
11109 Computational Linguistics.
- 11110 Galley, M., M. Hopkins, K. Knight, and D. Marcu (2004). What's in a translation rule? In
11111 *Proceedings of the North American Chapter of the Association for Computational Linguistics*
11112 (NAACL), pp. 273–280.
- 11113 Galley, M., K. R. McKeown, E. Fosler-Lussier, and H. Jing (2003). Discourse segmentation
11114 of multi-party conversation. In *Proceedings of the Association for Computational Linguistics*
11115 (ACL).
- 11116 Ganchev, K. and M. Dredze (2008). Small statistical models by random feature mixing. In
11117 *Proceedings of the ACL08 HLT Workshop on Mobile Language Processing*, pp. 19–20.
- 11118 Ganchev, K., J. Graça, J. Gillenwater, and B. Taskar (2010). Posterior regularization for
11119 structured latent variable models. *The Journal of Machine Learning Research* 11, 2001–
11120 2049.

- 11121 Ganin, Y., E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand,
11122 and V. Lempitsky (2016). Domain-adversarial training of neural networks. *Journal of
11123 Machine Learning Research* 17(59), 1–35.
- 11124 Gao, J., G. Andrew, M. Johnson, and K. Toutanova (2007). A comparative study of param-
11125 eter estimation methods for statistical natural language processing. In *Proceedings of the
11126 Association for Computational Linguistics (ACL)*, pp. 824–831.
- 11127 Gatt, A. and E. Krahmer (2018). Survey of the state of the art in natural language genera-
11128 tion: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research* 61,
11129 65–170.
- 11130 Gatt, A. and E. Reiter (2009). Simplenlg: A realisation engine for practical applications.
11131 In *Proceedings of the 12th European Workshop on Natural Language Generation*, pp. 90–93.
11132 Association for Computational Linguistics.
- 11133 Ge, D., X. Jiang, and Y. Ye (2011). A note on the complexity of $l_1 p$ minimization. *Mathe-
11134 matical programming* 129(2), 285–299.
- 11135 Ge, N., J. Hale, and E. Charniak (1998). A statistical approach to anaphora resolution. In
11136 *Proceedings of the sixth workshop on very large corpora*, Volume 71, pp. 76.
- 11137 Ge, R., F. Huang, C. Jin, and Y. Yuan (2015). Escaping from saddle points — online stochas-
11138 tic gradient for tensor decomposition. In P. Grünwald, E. Hazan, and S. Kale (Eds.),
11139 *Proceedings of the Conference On Learning Theory (COLT)*.
- 11140 Ge, R. and R. J. Mooney (2005). A statistical semantic parser that integrates syntax and
11141 semantics. In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp.
11142 9–16.
- 11143 Geach, P. T. (1962). *Reference and generality: An examination of some medieval and modern
11144 theories*. Cornell University Press.
- 11145 Gildea, D. and D. Jurafsky (2002). Automatic labeling of semantic roles. *Computational
11146 linguistics* 28(3), 245–288.
- 11147 Gimpel, K., N. Schneider, B. O’Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yo-
11148 gatama, J. Flanigan, and N. A. Smith (2011). Part-of-speech tagging for Twitter: an-
11149 notation, features, and experiments. In *Proceedings of the Association for Computational
11150 Linguistics (ACL)*, pp. 42–47.
- 11151 Glass, J., T. J. Hazen, S. Cyphers, I. Malioutov, D. Huynh, and R. Barzilay (2007). Recent
11152 progress in the mit spoken lecture processing project. In *Eighth Annual Conference of the
11153 International Speech Communication Association*.

- 11154 Glorot, X. and Y. Bengio (2010). Understanding the difficulty of training deep feedforward
 11155 neural networks. In *Proceedings of Artificial Intelligence and Statistics (AISTATS)*, pp. 249–
 11156 256.
- 11157 Glorot, X., A. Bordes, and Y. Bengio (2011). Deep sparse rectifier networks. In *Proceedings*
 11158 *of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP*
 11159 *Volume*, Volume 15, pp. 315–323.
- 11160 Godfrey, J. J., E. C. Holliman, and J. McDaniel (1992). Switchboard: Telephone speech
 11161 corpus for research and development. In *Proceedings of the International Conference on*
 11162 *Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 517–520. IEEE.
- 11163 Goldberg, Y. (2017a, June). An adversarial review of “adversarial generation of
 11164 natural language”. [https://medium.com/@yoav.goldberg/an-adversarial-review-of-](https://medium.com/@yoav.goldberg/an-adversarial-review-of-adversarial-generation-of-natural-language-409ac3378bd7)
 11165 [adversarial-generation-of-natural-language-409ac3378bd7](https://medium.com/@yoav.goldberg/an-adversarial-review-of-natural-language-409ac3378bd7).
- 11166 Goldberg, Y. (2017b). *Neural Network Methods for Natural Language Processing*. Synthesis
 11167 Lectures on Human Language Technologies. Morgan & Claypool Publishers.
- 11168 Goldberg, Y. and M. Elhadad (2010). An efficient algorithm for easy-first non-directional
 11169 dependency parsing. In *Proceedings of the North American Chapter of the Association for*
 11170 *Computational Linguistics (NAACL)*, pp. 742–750.
- 11171 Goldberg, Y. and J. Nivre (2012). A dynamic oracle for arc-eager dependency parsing.
 11172 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.
 11173 959–976.
- 11174 Goldberg, Y., K. Zhao, and L. Huang (2013). Efficient implementation of beam-search
 11175 incremental parsers. In *ACL (2)*, pp. 628–633.
- 11176 Goldwater, S. and T. Griffiths (2007). A fully bayesian approach to unsupervised part-of-
 11177 speech tagging. In *Annual meeting-association for computational linguistics*, Volume 45.
- 11178 Gonçalo Oliveira, H. R., F. A. Cardoso, and F. C. Pereira (2007). Tra-la-lyrics: An approach
 11179 to generate text based on rhythm. In *Proceedings of the 4th. International Joint Workshop*
 11180 *on Computational Creativity*. A. Cardoso and G. Wiggins.
- 11181 Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep learning*. MIT Press.
- 11182 Goodman, J. T. (2001). A bit of progress in language modeling. *Computer Speech & Lan-*
 11183 *guage* 15(4), 403–434.
- 11184 Gouws, S., D. Metzler, C. Cai, and E. Hovy (2011). Contextual bearing on linguistic varia-
 11185 *tion in social media*. In *LASM*.

- 11186 Goyal, A., H. Daume III, and S. Venkatasubramanian (2009). Streaming for large scale
11187 nlp: Language modeling. In *Proceedings of the North American Chapter of the Association*
11188 for Computational Linguistics (NAACL), pp. 512–520.
- 11189 Graves, A. (2012). Sequence transduction with recurrent neural networks. In *Proceedings*
11190 of the International Conference on Machine Learning (ICML).
- 11191 Graves, A. and N. Jaitly (2014). Towards end-to-end speech recognition with recur-
11192 rent neural networks. In *Proceedings of the International Conference on Machine Learning*
11193 (ICML), pp. 1764–1772.
- 11194 Graves, A. and J. Schmidhuber (2005). Framewise phoneme classification with bidirec-
11195 tional lstm and other neural network architectures. *Neural Networks* 18(5), 602–610.
- 11196 Grice, H. P. (1975). Logic and conversation. In P. Cole and J. L. Morgan (Eds.), *Syntax and*
11197 *Semantics Volume 3: Speech Acts*, pp. 41–58. Academic Press.
- 11198 Grishman, R. (2012). Information extraction: Capabilities and challenges. Notes prepared
11199 for the 2012 International Winter School in Language and Speech Technologies, Rovira
11200 i Virgili University, Tarragona, Spain.
- 11201 Grishman, R. (2015). Information extraction. *IEEE Intelligent Systems* 30(5), 8–15.
- 11202 Grishman, R., C. Macleod, and J. Sterling (1992). Evaluating parsing strategies using
11203 standardized parse files. In *Proceedings of the third conference on Applied natural language*
11204 *processing*, pp. 156–161. Association for Computational Linguistics.
- 11205 Grishman, R. and B. Sundheim (1996). Message understanding conference-6: A brief his-
11206 tory. In *Proceedings of the International Conference on Computational Linguistics (COLING)*,
11207 pp. 466–471.
- 11208 Groenendijk, J. and M. Stokhof (1991). Dynamic predicate logic. *Linguistics and philoso-*
11209 *phy* 14(1), 39–100.
- 11210 Grosz, B. J. (1979). Focusing and description in natural language dialogues. Technical
11211 report, SRI INTERNATIONAL MENLO PARK CA.
- 11212 Grosz, B. J., S. Weinstein, and A. K. Joshi (1995). Centering: A framework for modeling
11213 the local coherence of discourse. *Computational linguistics* 21(2), 203–225.
- 11214 Gu, J., Z. Lu, H. Li, and V. O. Li (2016). Incorporating copying mechanism in sequence-to-
11215 sequence learning. In *Proceedings of the Association for Computational Linguistics (ACL)*,
11216 pp. 1631–1640.
- 11217 Gulcehre, C., S. Ahn, R. Nallapati, B. Zhou, and Y. Bengio (2016). Pointing the unknown
11218 words. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 140–149.

- 11219 Gutmann, M. U. and A. Hyvärinen (2012). Noise-contrastive estimation of unnormalized
 11220 statistical models, with applications to natural image statistics. *The Journal of Machine
 11221 Learning Research* 13(1), 307–361.
- 11222 Haghghi, A. and D. Klein (2007). Unsupervised coreference resolution in a nonparametric
 11223 bayesian model. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11224 Haghghi, A. and D. Klein (2009). Simple coreference resolution with rich syntactic and
 11225 semantic features. In *Proceedings of Empirical Methods for Natural Language Processing
 11226 (EMNLP)*, pp. 1152–1161.
- 11227 Haghghi, A. and D. Klein (2010). Coreference resolution in a modular, entity-centered
 11228 model. In *Proceedings of the North American Chapter of the Association for Computational
 11229 Linguistics (NAACL)*, pp. 385–393.
- 11230 Hajič, J. and B. Hladká (1998). Tagging inflective languages: Prediction of morphological
 11231 categories for a rich, structured tagset. In *Proceedings of the Association for Computational
 11232 Linguistics (ACL)*, pp. 483–490.
- 11233 Halliday, M. and R. Hasan (1976). *Cohesion in English*. London: Longman.
- 11234 Hammerton, J. (2003). Named entity recognition with long short-term memory. In *Pro-
 11235 ceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 172–175.
- 11236 Han, X. and L. Sun (2012). An entity-topic model for entity linking. In *Proceedings of
 11237 Empirical Methods for Natural Language Processing (EMNLP)*, pp. 105–115.
- 11238 Han, X., L. Sun, and J. Zhao (2011). Collective entity linking in web text: a graph-based
 11239 method. In *Proceedings of ACM SIGIR conference on Research and development in informa-
 11240 tion retrieval*, pp. 765–774.
- 11241 Hannak, A., E. Anderson, L. F. Barrett, S. Lehmann, A. Mislove, and M. Riedewald (2012).
 11242 Tweetin'in the rain: Exploring societal-scale effects of weather on mood. In *Proceedings
 11243 of the International Conference on Web and Social Media (ICWSM)*.
- 11244 Hardmeier, C. (2012). Discourse in statistical machine translation. a survey and a case
 11245 study. *Discours. Revue de linguistique, psycholinguistique et informatique. A journal of lin-
 11246 guistics, psycholinguistics and computational linguistics* (11).
- 11247 Haspelmath, M. and A. Sims (2013). *Understanding morphology*. Routledge.
- 11248 Hastie, T., R. Tibshirani, and J. Friedman (2009). *The elements of statistical learning* (Second
 11249 ed.). New York: Springer.

- 11250 Hatzivassiloglou, V. and K. R. McKeown (1997). Predicting the semantic orientation of
11251 adjectives. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 174–
11252 181.
- 11253 Hayes, A. F. and K. Krippendorff (2007). Answering the call for a standard reliability
11254 measure for coding data. *Communication methods and measures* 1(1), 77–89.
- 11255 He, H., A. Balakrishnan, M. Eric, and P. Liang (2017). Learning symmetric collaborative
11256 dialogue agents with dynamic knowledge graph embeddings. In *Proceedings of the As-
11257 sociation for Computational Linguistics (ACL)*, pp. 1766–1776.
- 11258 He, K., X. Zhang, S. Ren, and J. Sun (2015). Delving deep into rectifiers: Surpassing
11259 human-level performance on imagenet classification. In *Proceedings of the International
11260 Conference on Computer Vision (ICCV)*, pp. 1026–1034.
- 11261 He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep residual learning for image recognition.
11262 In *Proceedings of the International Conference on Computer Vision (ICCV)*, pp. 770–778.
- 11263 He, L., K. Lee, M. Lewis, and L. Zettlemoyer (2017). Deep semantic role labeling: What
11264 works and what's next. In *Proceedings of the Association for Computational Linguistics
11265 (ACL)*.
- 11266 He, Z., S. Liu, M. Li, M. Zhou, L. Zhang, and H. Wang (2013). Learning entity repre-
11267 sentation for entity disambiguation. In *Proceedings of the Association for Computational
11268 Linguistics (ACL)*, pp. 30–34.
- 11269 Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. In
11270 *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp. 539–
11271 545. Association for Computational Linguistics.
- 11272 Hearst, M. A. (1997). Texttiling: Segmenting text into multi-paragraph subtopic passages.
11273 *Computational linguistics* 23(1), 33–64.
- 11274 Heerschap, B., F. Goossen, A. Hogenboom, F. Frasincar, U. Kaymak, and F. de Jong (2011).
11275 Polarity analysis of texts using discourse structure. In *Proceedings of the 20th ACM inter-
11276 national conference on Information and knowledge management*, pp. 1061–1070. ACM.
- 11277 Henderson, J. (2004). Discriminative training of a neural network statistical parser. In
11278 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 95–102.
- 11279 Hendrickx, I., S. N. Kim, Z. Kozareva, P. Nakov, D. Ó Séaghdha, S. Padó, M. Pennacchiotti,
11280 L. Romano, and S. Szpakowicz (2009). SemEval-2010 task 8: Multi-way classification of
11281 semantic relations between pairs of nominals. In *Proceedings of the Workshop on Semantic
11282 Evaluations: Recent Achievements and Future Directions*, pp. 94–99. Association for Com-
11283 putational Linguistics.

- 11284 Hermann, K. M., T. Kočiský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and
 11285 P. Blunsom (2015). Teaching machines to read and comprehend. In *Advances in Neu-*
 11286 *ral Information Processing Systems*, pp. 1693–1701.
- 11287 Hernault, H., H. Prendinger, D. A. duVerle, and M. Ishizuka (2010). HILDA: A discourse
 11288 parser using support vector machine classification. *Dialogue and Discourse* 1(3), 1–33.
- 11289 Hill, F., A. Bordes, S. Chopra, and J. Weston (2016). The goldilocks principle: Reading
 11290 children’s books with explicit memory representations. In *Proceedings of the International*
 11291 *Conference on Learning Representations (ICLR)*.
- 11292 Hill, F., K. Cho, and A. Korhonen (2016). Learning distributed representations of sentences
 11293 from unlabelled data. In *Proceedings of the North American Chapter of the Association for*
 11294 *Computational Linguistics (NAACL)*.
- 11295 Hindle, D. and M. Rooth (1993). Structural ambiguity and lexical relations. *Computational*
 11296 *linguistics* 19(1), 103–120.
- 11297 Hirao, T., Y. Yoshida, M. Nishino, N. Yasuda, and M. Nagata (2013). Single-document
 11298 summarization as a tree knapsack problem. In *Proceedings of Empirical Methods for Nat-*
 11299 *ural Language Processing (EMNLP)*, pp. 1515–1520.
- 11300 Hirschman, L. and R. Gaizauskas (2001). Natural language question answering: the view
 11301 from here. *natural language engineering* 7(4), 275–300.
- 11302 Hirschman, L., M. Light, E. Breck, and J. D. Burger (1999). Deep read: A reading compre-
 11303 hension system. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
 11304 325–332.
- 11305 Hobbs, J. R. (1978). Resolving pronoun references. *Lingua* 44(4), 311–338.
- 11306 Hobbs, J. R., D. Appelt, J. Bear, D. Israel, M. Kameyama, M. Stickel, and M. Tyson (1997).
 11307 Fastus: A cascaded finite-state transducer for extracting information from natural-
 11308 language text. *Finite-state language processing*, 383–406.
- 11309 Hochreiter, S. and J. Schmidhuber (1997). Long short-term memory. *Neural computa-*
 11310 *tion* 9(8), 1735–1780.
- 11311 Hockenmaier, J. and M. Steedman (2007). Ccgbank: a corpus of ccg derivations and de-
 11312 pendency structures extracted from the penn treebank. *Computational Linguistics* 33(3),
 11313 355–396.
- 11314 Hoffart, J., M. A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, B. Taneva,
 11315 S. Thater, and G. Weikum (2011). Robust disambiguation of named entities in text. In
 11316 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 782–792.

- 11317 Hoffmann, R., C. Zhang, X. Ling, L. Zettlemoyer, and D. S. Weld (2011). Knowledge-based
11318 weak supervision for information extraction of overlapping relations. In *Proceedings of
11319 the Association for Computational Linguistics (ACL)*, pp. 541–550.
- 11320 Holmstrom, L. and P. Koistinen (1992). Using additive noise in back-propagation training.
11321 *IEEE Transactions on Neural Networks* 3(1), 24–38.
- 11322 Hovy, E. and J. Lavid (2010). Towards a ‘science’ of corpus annotation: a new method-
11323 ological challenge for corpus linguistics. *International journal of translation* 22(1), 13–36.
- 11324 Hsu, D., S. M. Kakade, and T. Zhang (2012). A spectral algorithm for learning hidden
11325 markov models. *Journal of Computer and System Sciences* 78(5), 1460–1480.
- 11326 Hu, M. and B. Liu (2004). Mining and summarizing customer reviews. In *Proceedings of
11327 Knowledge Discovery and Data Mining (KDD)*, pp. 168–177.
- 11328 Hu, Z., Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing (2017). Toward controlled
11329 generation of text. In *International Conference on Machine Learning*, pp. 1587–1596.
- 11330 Huang, F. and A. Yates (2012). Biased representation learning for domain adaptation. In
11331 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1313–1323.
- 11332 Huang, L. and D. Chiang (2007). Forest rescoring: Faster decoding with integrated lan-
11333 guage models. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
11334 144–151.
- 11335 Huang, L., S. Fayong, and Y. Guo (2012). Structured perceptron with inexact search. In
11336 *Proceedings of the North American Chapter of the Association for Computational Linguistics
11337 (NAACL)*, pp. 142–151.
- 11338 Huang, Y. (2015). *Pragmatics* (Second ed.). Oxford Textbooks in Linguistics. Oxford Uni-
11339 versity Press.
- 11340 Huang, Z., W. Xu, and K. Yu (2015). Bidirectional lstm-crf models for sequence tagging.
11341 *arXiv preprint arXiv:1508.01991*.
- 11342 Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes.
11343 *Proceedings of the IRE* 40(9), 1098–1101.
- 11344 Humphreys, K., R. Gaizauskas, and S. Azzam (1997). Event coreference for information
11345 extraction. In *Proceedings of a Workshop on Operational Factors in Practical, Robust Anaphora
11346 Resolution for Unrestricted Texts*, pp. 75–81. Association for Computational Linguistics.
- 11347 Ide, N. and Y. Wilks (2006). Making sense about sense. In *Word sense disambiguation*, pp.
11348 47–73. Springer.

- 11349 Ioffe, S. and C. Szegedy (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 448–456.
- 11352 Isozaki, H., T. Hirao, K. Duh, K. Sudoh, and H. Tsukada (2010). Automatic evaluation of translation quality for distant language pairs. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 944–952.
- 11355 Iyyer, M., V. Manjunatha, J. Boyd-Graber, and H. Daumé III (2015). Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1681–1691.
- 11358 James, G., D. Witten, T. Hastie, and R. Tibshirani (2013). *An introduction to statistical learning*, Volume 112. Springer.
- 11360 Janin, A., D. Baron, J. Edwards, D. Ellis, D. Gelbart, N. Morgan, B. Peskin, T. Pfau, E. Shriberg, A. Stolcke, et al. (2003). The ICSI meeting corpus. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, Volume 1, pp. I–I. IEEE.
- 11364 Jean, S., K. Cho, R. Memisevic, and Y. Bengio (2015). On using very large target vocabulary for neural machine translation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1–10.
- 11367 Jeong, M., C.-Y. Lin, and G. G. Lee (2009). Semi-supervised speech act recognition in emails and forums. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1250–1259.
- 11370 Ji, H. and R. Grishman (2011). Knowledge base population: Successful approaches and challenges. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1148–1158.
- 11373 Ji, Y., T. Cohn, L. Kong, C. Dyer, and J. Eisenstein (2015). Document context language models. In *International Conference on Learning Representations, Workshop Track*, Volume abs/1511.03962.
- 11376 Ji, Y. and J. Eisenstein (2014). Representation learning for text-level discourse parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11378 Ji, Y. and J. Eisenstein (2015, June). One vector is not enough: Entity-augmented distributional semantics for discourse relations. *Transactions of the Association for Computational Linguistics (TACL)*.

- 11381 Ji, Y., G. Haffari, and J. Eisenstein (2016). A latent variable recurrent neural network for
11382 discourse relation language models. In *Proceedings of the North American Chapter of the*
11383 *Association for Computational Linguistics (NAACL)*.
- 11384 Ji, Y. and N. A. Smith (2017). Neural discourse structure for text categorization. In *Pro-
11385 ceedings of the Association for Computational Linguistics (ACL)*, pp. 996–1005.
- 11386 Ji, Y., C. Tan, S. Martschat, Y. Choi, and N. A. Smith (2017). Dynamic entity representations
11387 in neural language models. In *Proceedings of Empirical Methods for Natural Language
11388 Processing (EMNLP)*, pp. 1831–1840.
- 11389 Jiang, L., M. Yu, M. Zhou, X. Liu, and T. Zhao (2011). Target-dependent twitter sentiment
11390 classification. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
11391 151–160.
- 11392 Jing, H. (2000). Sentence reduction for automatic text summarization. In *Proceedings of
11393 the sixth conference on Applied natural language processing*, pp. 310–315. Association for
11394 Computational Linguistics.
- 11395 Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of
11396 Knowledge Discovery and Data Mining (KDD)*, pp. 133–142.
- 11397 Jockers, M. L. (2015). Szuzhet? <http://bla.bla.com>.
- 11398 Johnson, A. E., T. J. Pollard, L. Shen, H. L. Li-wei, M. Feng, M. Ghassemi, B. Moody,
11399 P. Szolovits, L. A. Celi, and R. G. Mark (2016). Mimic-iii, a freely accessible critical care
11400 database. *Scientific data* 3, 160035.
- 11401 Johnson, M. (1998). Pcfg models of linguistic tree representations. *Computational Linguis-
11402 tics* 24(4), 613–632.
- 11403 Johnson, R. and T. Zhang (2017). Deep pyramid convolutional neural networks for text
11404 categorization. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
11405 562–570.
- 11406 Joshi, A. K. (1985). How much context-sensitivity is required to provide reasonable struc-
11407 tural descriptions? – tree adjoining grammars. In *Natural Language Processing – Theoret-
11408 ical, Computational and Psychological Perspective*. New York, NY: Cambridge University
11409 Press.
- 11410 Joshi, A. K. and Y. Schabes (1997). Tree-adjoining grammars. In *Handbook of formal lan-
11411 guages*, pp. 69–123. Springer.
- 11412 Joshi, A. K., K. V. Shanker, and D. Weir (1991). The convergence of mildly context-sensitive
11413 grammar formalisms. In *Foundational Issues in Natural Language Processing*. Cambridge
11414 MA: MIT Press.

- 11415 Jozefowicz, R., O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu (2016). Exploring the limits
11416 of language modeling. *arXiv preprint arXiv:1602.02410*.
- 11417 Jozefowicz, R., W. Zaremba, and I. Sutskever (2015). An empirical exploration of recurrent
11418 network architectures. In *Proceedings of the International Conference on Machine Learning*
11419 (*ICML*), pp. 2342–2350.
- 11420 Jurafsky, D. (1996). A probabilistic model of lexical and syntactic access and disambiguation.
11421 *Cognitive Science* 20(2), 137–194.
- 11422 Jurafsky, D. and J. H. Martin (2009). *Speech and Language Processing* (Second ed.). Prentice
11423 Hall.
- 11424 Jurafsky, D. and J. H. Martin (2018). *Speech and Language Processing* (Third ed.). Prentice
11425 Hall.
- 11426 Kadlec, R., M. Schmid, O. Bajgar, and J. Kleindienst (2016). Text understanding with
11427 the attention sum reader network. In *Proceedings of the Association for Computational*
11428 *Linguistics (ACL)*, pp. 908–918.
- 11429 Kalchbrenner, N. and P. Blunsom (2013, August). Recurrent convolutional neural net-
11430 works for discourse compositionality. In *Proceedings of the Workshop on Continuous Vec-*
11431 *tor Space Models and their Compositionality*, Sofia, Bulgaria, pp. 119–126. Association for
11432 Computational Linguistics.
- 11433 Kalchbrenner, N., E. Grefenstette, and P. Blunsom (2014). A convolutional neural network
11434 for modelling sentences. In *Proceedings of the Association for Computational Linguistics*
11435 (*ACL*), pp. 655–665.
- 11436 Karlsson, F. (2007). Constraints on multiple center-embedding of clauses. *Journal of Lin-*
11437 *guistics* 43(02), 365–392.
- 11438 Kate, R. J., Y. W. Wong, and R. J. Mooney (2005). Learning to transform natural to formal
11439 languages. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.
- 11440 Kehler, A. (2007). Rethinking the SMASH approach to pronoun interpretation. In *Interdis-*
11441 *ciplinary perspectives on reference processing*, New Directions in Cognitive Science Series,
11442 pp. 95–122. Oxford University Press.
- 11443 Kibble, R. and R. Power (2004). Optimizing referential coherence in text generation. *Com-*
11444 *putational Linguistics* 30(4), 401–416.
- 11445 Kilgarriff, A. (1997). I don't believe in word senses. *Computers and the Humanities* 31(2),
11446 91–113.

- 11447 Kilgarriff, A. and G. Grefenstette (2003). Introduction to the special issue on the web as
11448 corpus. *Computational linguistics* 29(3), 333–347.
- 11449 Kim, M.-J. (2002). Does korean have adjectives? *MIT Working Papers in Linguistics* 43,
11450 71–89.
- 11451 Kim, S.-M. and E. Hovy (2006, July). Extracting opinions, opinion holders, and topics
11452 expressed in online news media text. In *Proceedings of the Workshop on Sentiment and*
11453 *Subjectivity in Text*, Sydney, Australia, pp. 1–8. Association for Computational Linguis-
11454 tics.
- 11455 Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings*
11456 *of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1746–1751.
- 11457 Kim, Y., C. Denton, L. Hoang, and A. M. Rush (2017). Structured attention networks. In
11458 *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 11459 Kim, Y., Y. Jernite, D. Sontag, and A. M. Rush (2016). Character-aware neural language
11460 models. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.
- 11461 Kingma, D. and J. Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint*
11462 *arXiv:1412.6980*.
- 11463 Kiperwasser, E. and Y. Goldberg (2016). Simple and accurate dependency parsing using
11464 bidirectional lstm feature representations. *Transactions of the Association for Compu-
11465 tational Linguistics* 4, 313–327.
- 11466 Kipper-Schuler, K. (2005). *VerbNet: A broad-coverage, comprehensive verb lexicon*. Ph. D.
11467 thesis, Computer and Information Science, University of Pennsylvania.
- 11468 Kiros, R., R. Salakhutdinov, and R. Zemel (2014). Multimodal neural language models. In
11469 *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 595–603.
- 11470 Kiros, R., Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler
11471 (2015). Skip-thought vectors. In *Neural Information Processing Systems (NIPS)*.
- 11472 Klein, D. and C. D. Manning (2003). Accurate unlexicalized parsing. In *Proceedings of the*
11473 *Association for Computational Linguistics (ACL)*, pp. 423–430.
- 11474 Klein, D. and C. D. Manning (2004). Corpus-based induction of syntactic structure: Mod-
11475 els of dependency and constituency. In *Proceedings of the Association for Computational*
11476 *Linguistics (ACL)*.
- 11477 Klein, G., Y. Kim, Y. Deng, J. Senellart, and A. M. Rush (2017). Opennmt: Open-source
11478 toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*.

- 11479 Klementiev, A., I. Titov, and B. Bhattachari (2012). Inducing crosslingual distributed representations of words. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp. 1459–1474.
- 11482 Klenner, M. (2007). Enforcing consistency on coreference sets. In *Recent Advances in Natural Language Processing (RANLP)*, pp. 323–328.
- 11484 Knight, K. (1999). Decoding complexity in word-replacement translation models. *Computational Linguistics* 25(4), 607–615.
- 11486 Knight, K. and J. Graehl (1998). Machine transliteration. *Computational Linguistics* 24(4), 599–612.
- 11488 Knight, K. and D. Marcu (2000). Statistics-based summarization-step one: Sentence compression. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 703–710.
- 11491 Knight, K. and J. May (2009). Applications of weighted automata in natural language processing. In *Handbook of Weighted Automata*, pp. 571–596. Springer.
- 11493 Knott, A. (1996). *A data-driven methodology for motivating a set of coherence relations*. Ph. D. thesis, The University of Edinburgh.
- 11495 Koehn, P. (2005). Europarl: A parallel corpus for statistical machine translation. In *MT summit*, Volume 5, pp. 79–86.
- 11497 Koehn, P. (2009). *Statistical machine translation*. Cambridge University Press.
- 11498 Koehn, P. (2017). Neural machine translation. *arXiv preprint arXiv:1709.07809*.
- 11499 Konstas, I. and M. Lapata (2013). A global model for concept-to-text generation. *Journal of Artificial Intelligence Research* 48, 305–346.
- 11501 Koo, T., X. Carreras, and M. Collins (2008, jun). Simple semi-supervised dependency parsing. In *Proceedings of ACL-08: HLT*, Columbus, Ohio, pp. 595–603. Association for Computational Linguistics.
- 11504 Koo, T. and M. Collins (2005). Hidden-variable models for discriminative reranking. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 507–514.
- 11506 Koo, T. and M. Collins (2010). Efficient third-order dependency parsers. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11508 Koo, T., A. Globerson, X. Carreras, and M. Collins (2007). Structured prediction models via the matrix-tree theorem. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 141–150.

- 11511 Kovach, B. and T. Rosenstiel (2014). *The elements of journalism: What newspeople should know
11512 and the public should expect*. Three Rivers Press.
- 11513 Krishnamurthy, J. (2016). Probabilistic models for learning a semantic parser lexicon. In
11514 *Proceedings of the North American Chapter of the Association for Computational Linguistics
11515 (NAACL)*, pp. 606–616.
- 11516 Krishnamurthy, J. and T. M. Mitchell (2012). Weakly supervised training of semantic
11517 parsers. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
11518 pp. 754–765.
- 11519 Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep
11520 convolutional neural networks. In *Neural Information Processing Systems (NIPS)*, pp.
11521 1097–1105.
- 11522 Kübler, S., R. McDonald, and J. Nivre (2009). Dependency parsing. *Synthesis Lectures on
11523 Human Language Technologies* 1(1), 1–127.
- 11524 Kuhlmann, M. and J. Nivre (2010). Transition-based techniques for non-projective depen-
11525 dency parsing. *Northern European Journal of Language Technology (NEJLT)* 2(1), 1–19.
- 11526 Kummerfeld, J. K., T. Berg-Kirkpatrick, and D. Klein (2015). An empirical analysis of op-
11527 timization for max-margin NLP. In *Proceedings of Empirical Methods for Natural Language
11528 Processing (EMNLP)*.
- 11529 Kwiatkowski, T., S. Goldwater, L. Zettlemoyer, and M. Steedman (2012). A probabilistic
11530 model of syntactic and semantic acquisition from child-directed utterances and their
11531 meanings. In *Proceedings of the European Chapter of the Association for Computational Lin-
11532 guistics (EACL)*, pp. 234–244.
- 11533 Lafferty, J., A. McCallum, and F. Pereira (2001). Conditional random fields: Probabilistic
11534 models for segmenting and labeling sequence data. In *icml*.
- 11535 Lakoff, G. (1973). Hedges: A study in meaning criteria and the logic of fuzzy concepts.
11536 *Journal of philosophical logic* 2(4), 458–508.
- 11537 Lample, G., M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer (2016). Neural
11538 architectures for named entity recognition. In *Proceedings of the North American Chapter
11539 of the Association for Computational Linguistics (NAACL)*, pp. 260–270.
- 11540 Langkilde, I. and K. Knight (1998). Generation that exploits corpus-based statistical
11541 knowledge. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 704–
11542 710.

- 11543 Lapata, M. (2003). Probabilistic text structuring: Experiments with sentence ordering. In
 11544 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 545–552.
- 11545 Lappin, S. and H. J. Leass (1994). An algorithm for pronominal anaphora resolution.
 11546 *Computational linguistics* 20(4), 535–561.
- 11547 Lari, K. and S. J. Young (1990). The estimation of stochastic context-free grammars using
 11548 the inside-outside algorithm. *Computer speech & language* 4(1), 35–56.
- 11549 Lascarides, A. and N. Asher (2007). Segmented discourse representation theory: Dynamic
 11550 semantics with discourse structure. In *Computing meaning*, pp. 87–124. Springer.
- 11551 Law, E. and L. v. Ahn (2011). Human computation. *Synthesis Lectures on Artificial Intelli-*
 11552 *gence and Machine Learning* 5(3), 1–121.
- 11553 Lebret, R., D. Grangier, and M. Auli (2016). Neural text generation from structured data
 11554 with application to the biography domain. In *Proceedings of Empirical Methods for Natural*
 11555 *Language Processing (EMNLP)*, pp. 1203–1213.
- 11556 LeCun, Y. and Y. Bengio (1995). Convolutional networks for images, speech, and time
 11557 series. *The handbook of brain theory and neural networks* 3361.
- 11558 LeCun, Y., L. Bottou, G. B. Orr, and K.-R. Müller (1998). Efficient backprop. In *Neural*
 11559 *networks: Tricks of the trade*, pp. 9–50. Springer.
- 11560 Lee, C. M. and S. S. Narayanan (2005). Toward detecting emotions in spoken dialogs.
 11561 *IEEE transactions on speech and audio processing* 13(2), 293–303.
- 11562 Lee, H., A. Chang, Y. Peirsman, N. Chambers, M. Surdeanu, and D. Jurafsky (2013). De-
 11563 terministic coreference resolution based on entity-centric, precision-ranked rules. *Com-*
 11564 *putational Linguistics* 39(4), 885–916.
- 11565 Lee, H., Y. Peirsman, A. Chang, N. Chambers, M. Surdeanu, and D. Jurafsky (2011). Stan-
 11566 ford’s multi-pass sieve coreference resolution system at the conll-2011 shared task. In
 11567 *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 28–34. Associa-
 11568 tion for Computational Linguistics.
- 11569 Lee, K., L. He, M. Lewis, and L. Zettlemoyer (2017). End-to-end neural coreference reso-
 11570 lution. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11571 Lenat, D. B., R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd (1990). Cyc: toward
 11572 programs with common sense. *Communications of the ACM* 33(8), 30–49.
- 11573 Lesk, M. (1986). Automatic sense disambiguation using machine readable dictionaries:
 11574 how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual interna-*
 11575 *tional conference on Systems documentation*, pp. 24–26. ACM.

- 11576 Levesque, H. J., E. Davis, and L. Morgenstern (2011). The winograd schema challenge.
11577 In *Aaa spring symposium: Logical formalizations of commonsense reasoning*, Volume 46, pp.
11578 47.
- 11579 Levin, E., R. Pieraccini, and W. Eckert (1998). Using markov decision process for learning
11580 dialogue strategies. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the*
11581 *1998 IEEE International Conference on*, Volume 1, pp. 201–204. IEEE.
- 11582 Levy, O. and Y. Goldberg (2014). Dependency-based word embeddings. In *Proceedings of*
11583 *the Association for Computational Linguistics (ACL)*, pp. 302–308.
- 11584 Levy, O., Y. Goldberg, and I. Dagan (2015). Improving distributional similarity with
11585 lessons learned from word embeddings. *Transactions of the Association for Computational*
11586 *Linguistics* 3, 211–225.
- 11587 Levy, R. and C. Manning (2009). An informal introduction to computational semantics.
- 11588 Lewis, M. and M. Steedman (2013). Combined distributional and logical semantics. *Trans-*
11589 *actions of the Association for Computational Linguistics* 1, 179–192.
- 11590 Lewis II, P. M. and R. E. Stearns (1968). Syntax-directed transduction. *Journal of the ACM*
11591 (*JACM*) 15(3), 465–488.
- 11592 Li, J. and D. Jurafsky (2015). Do multi-sense embeddings improve natural language
11593 understanding? In *Proceedings of Empirical Methods for Natural Language Processing*
11594 (*EMNLP*), pp. 1722–1732.
- 11595 Li, J. and D. Jurafsky (2017). Neural net models of open-domain discourse coherence. In
11596 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 198–209.
- 11597 Li, J., R. Li, and E. Hovy (2014). Recursive deep models for discourse parsing. In *Proceed-*
11598 *ings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11599 Li, J., M.-T. Luong, and D. Jurafsky (2015). A hierarchical neural autoencoder for para-
11600 graphs and documents. In *Proceedings of Empirical Methods for Natural Language Process-*
11601 *ing (EMNLP)*.
- 11602 Li, J., T. Luong, D. Jurafsky, and E. Hovy (2015). When are tree structures necessary
11603 for deep learning of representations? In *Proceedings of Empirical Methods for Natural*
11604 *Language Processing (EMNLP)*, pp. 2304–2314.
- 11605 Li, J., W. Monroe, A. Ritter, D. Jurafsky, M. Galley, and J. Gao (2016, November). Deep
11606 reinforcement learning for dialogue generation. In *Proceedings of the 2016 Conference on*
11607 *Empirical Methods in Natural Language Processing*, Austin, Texas, pp. 1192–1202. Associa-
11608 *tion for Computational Linguistics*.

- 11609 Li, Q., S. Anzaroot, W.-P. Lin, X. Li, and H. Ji (2011). Joint inference for cross-document
 11610 information extraction. In *Proceedings of the International Conference on Information and*
 11611 *Knowledge Management (CIKM)*, pp. 2225–2228.
- 11612 Li, Q., H. Ji, and L. Huang (2013). Joint event extraction via structured prediction with
 11613 global features. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
 11614 73–82.
- 11615 Liang, P., A. Bouchard-Côté, D. Klein, and B. Taskar (2006). An end-to-end discriminative
 11616 approach to machine translation. In *Proceedings of the Association for Computational*
 11617 *Linguistics (ACL)*, pp. 761–768.
- 11618 Liang, P., M. Jordan, and D. Klein (2009). Learning semantic correspondences with less
 11619 supervision. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 91–
 11620 99.
- 11621 Liang, P., M. I. Jordan, and D. Klein (2013). Learning dependency-based compositional
 11622 semantics. *Computational Linguistics* 39(2), 389–446.
- 11623 Liang, P. and D. Klein (2009). Online em for unsupervised models. In *Proceedings of the*
 11624 *North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 611–
 11625 619.
- 11626 Liang, P., S. Petrov, M. I. Jordan, and D. Klein (2007). The infinite pcfg using hierarchical
 11627 dirichlet processes. In *Proceedings of Empirical Methods for Natural Language Processing*
 11628 (*EMNLP*), pp. 688–697.
- 11629 Liang, P. and C. Potts (2015). Bringing machine learning and compositional semantics
 11630 together. *Annual Review of Linguistics* 1(1), 355–376.
- 11631 Lieber, R. (2015). *Introducing morphology*. Cambridge University Press.
- 11632 Lin, D. (1998). Automatic retrieval and clustering of similar words. In *Proceedings of the*
 11633 *17th international conference on Computational linguistics-Volume 2*, pp. 768–774. Association
 11634 for Computational Linguistics.
- 11635 Lin, J. and C. Dyer (2010). Data-intensive text processing with mapreduce. *Synthesis*
 11636 *Lectures on Human Language Technologies* 3(1), 1–177.
- 11637 Lin, Z., M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio (2017). A
 11638 structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*.
- 11639 Lin, Z., M.-Y. Kan, and H. T. Ng (2009). Recognizing implicit discourse relations in the
 11640 penn discourse treebank. In *Proceedings of Empirical Methods for Natural Language Pro-*
 11641 *cessing (EMNLP)*, pp. 343–351.

- 11642 Lin, Z., H. T. Ng, and M.-Y. Kan (2011). Automatically evaluating text coherence using
11643 discourse relations. In *Proceedings of the Association for Computational Linguistics (ACL)*,
11644 pp. 997–1006.
- 11645 Lin, Z., H. T. Ng, and M. Y. Kan (2014, nov). A PDTB-styled end-to-end discourse parser.
11646 *Natural Language Engineering FirstView*, 1–34.
- 11647 Ling, W., C. Dyer, A. Black, and I. Trancoso (2015). Two/too simple adaptations of
11648 word2vec for syntax problems. In *Proceedings of the North American Chapter of the As-*
11649 *sociation for Computational Linguistics (NAACL)*.
- 11650 Ling, W., T. Luís, L. Marujo, R. F. Astudillo, S. Amir, C. Dyer, A. W. Black, and I. Trancoso
11651 (2015). Finding function in form: Compositional character models for open vocabulary
11652 word representation. In *Proceedings of Empirical Methods for Natural Language Processing*
11653 (*EMNLP*).
- 11654 Ling, W., G. Xiang, C. Dyer, A. Black, and I. Trancoso (2013). Microblogs as parallel cor-
11655 pora. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11656 Ling, X., S. Singh, and D. S. Weld (2015). Design challenges for entity linking. *Transactions*
11657 *of the Association for Computational Linguistics* 3, 315–328.
- 11658 Linguistic Data Consortium (2005, July). ACE (automatic content extraction) English an-
11659 notation guidelines for relations. Technical Report Version 5.8.3, Linguistic Data Con-
11660 sortium.
- 11661 Liu, B. (2015). *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. Cambridge
11662 University Press.
- 11663 Liu, D. C. and J. Nocedal (1989). On the limited memory BFGS method for large scale
11664 optimization. *Mathematical programming* 45(1-3), 503–528.
- 11665 Liu, Y., Q. Liu, and S. Lin (2006). Tree-to-string alignment template for statistical machine
11666 translation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 609–
11667 616.
- 11668 Loper, E. and S. Bird (2002). Nltk: The natural language toolkit. In *Proceedings of the ACL-*
11669 *02 Workshop on Effective tools and methodologies for teaching natural language processing and*
11670 *computational linguistics-Volume 1*, pp. 63–70. Association for Computational Linguistics.
- 11671 Louis, A., A. Joshi, and A. Nenkova (2010). Discourse indicators for content selection in
11672 summarization. In *Proceedings of the 11th Annual Meeting of the Special Interest Group on*
11673 *Discourse and Dialogue*, pp. 147–156. Association for Computational Linguistics.

- 11674 Louis, A. and A. Nenkova (2013). What makes writing great? first experiments on article
 11675 quality prediction in the science journalism domain. *Transactions of the Association for
 11676 Computational Linguistics* 1, 341–352.
- 11677 Loveland, D. W. (2016). *Automated Theorem Proving: a logical basis*. Elsevier.
- 11678 Lowe, R., N. Pow, I. V. Serban, and J. Pineau (2015). The ubuntu dialogue corpus: A large
 11679 dataset for research in unstructured multi-turn dialogue systems. In *Proceedings of the
 11680 Special Interest Group on Discourse and Dialogue (SIGDIAL)*.
- 11681 Luo, X. (2005). On coreference resolution performance metrics. In *Proceedings of Empirical
 11682 Methods for Natural Language Processing (EMNLP)*, pp. 25–32.
- 11683 Luo, X., A. Ittycheriah, H. Jing, N. Kambhatla, and S. Roukos (2004). A mention-
 11684 synchronous coreference resolution algorithm based on the bell tree. In *Proceedings
 11685 of the Association for Computational Linguistics (ACL)*.
- 11686 Luong, M.-T., R. Socher, and C. D. Manning (2013). Better word representations with
 11687 recursive neural networks for morphology. *CoNLL-2013* 104.
- 11688 Luong, T., H. Pham, and C. D. Manning (2015). Effective approaches to attention-based
 11689 neural machine translation. In *Proceedings of Empirical Methods for Natural Language
 11690 Processing (EMNLP)*, pp. 1412–1421.
- 11691 Luong, T., I. Sutskever, Q. Le, O. Vinyals, and W. Zaremba (2015). Addressing the rare
 11692 word problem in neural machine translation. In *Proceedings of the Association for Compu-
 11693 tational Linguistics (ACL)*, pp. 11–19.
- 11694 Maas, A. L., A. Y. Hannun, and A. Y. Ng (2013). Rectifier nonlinearities improve neu-
 11695 ral network acoustic models. In *Proceedings of the International Conference on Machine
 11696 Learning (ICML)*.
- 11697 Mairesse, F. and M. A. Walker (2011). Controlling user perceptions of linguistic style:
 11698 Trainable generation of personality traits. *Computational Linguistics* 37(3), 455–488.
- 11699 Mani, I., M. Verhagen, B. Wellner, C. M. Lee, and J. Pustejovsky (2006). Machine learning
 11700 of temporal relations. In *Proceedings of the Association for Computational Linguistics (ACL)*,
 11701 pp. 753–760.
- 11702 Mann, W. C. and S. A. Thompson (1988). Rhetorical structure theory: Toward a functional
 11703 theory of text organization. *Text* 8(3), 243–281.
- 11704 Manning, C. D. (2015). Computational linguistics and deep learning. *Computational Lin-
 11705 guistics* 41(4), 701–707.

- 11706 Manning, C. D. (2016). Computational linguistics and deep learning. *Computational Linguistics* 41(4).
- 11707
- 11708 Manning, C. D., P. Raghavan, H. Schütze, et al. (2008). *Introduction to information retrieval*, Volume 1. Cambridge university press.
- 11709
- 11710 Manning, C. D. and H. Schütze (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts: MIT press.
- 11711
- 11712 Marcu, D. (1996). Building up rhetorical structure trees. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 1069–1074.
- 11713
- 11714 Marcu, D. (1997a). From discourse structures to text summaries. In *Proceedings of the workshop on Intelligent Scalable Text Summarization*.
- 11715
- 11716 Marcu, D. (1997b). From local to global coherence: A bottom-up approach to text planning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 629–635.
- 11717
- 11718 Marcus, M. P., M. A. Marcinkiewicz, and B. Santorini (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics* 19(2), 313–330.
- 11719
- 11720 Maron, O. and T. Lozano-Pérez (1998). A framework for multiple-instance learning. In *Neural Information Processing Systems (NIPS)*, pp. 570–576.
- 11721
- 11722 Márquez, G. G. (1970). *One Hundred Years of Solitude*. Harper & Row. English translation by Gregory Rabassa.
- 11723
- 11724 Martins, A. F. T., N. A. Smith, and E. P. Xing (2009). Concise integer linear programming formulations for dependency parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 342–350.
- 11725
- 11726
- 11727 Martins, A. F. T., N. A. Smith, E. P. Xing, P. M. Q. Aguiar, and M. A. T. Figueiredo (2010). Turbo parsers: Dependency parsing by approximate variational inference. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 34–44.
- 11728
- 11729
- 11730 Matsuzaki, T., Y. Miyao, and J. Tsujii (2005). Probabilistic cfg with latent annotations. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 75–82.
- 11731
- 11732 Matthiessen, C. and J. A. Bateman (1991). *Text generation and systemic-functional linguistics: experiences from English and Japanese*. Pinter Publishers.
- 11733
- 11734 McCallum, A. and W. Li (2003). Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 188–191.
- 11735
- 11736
- 11737

- 11738 McCallum, A. and B. Wellner (2004). Conditional models of identity uncertainty with
 11739 application to noun coreference. In *NIPS*, pp. 905–912.
- 11740 McDonald, R., K. Crammer, and F. Pereira (2005). Online large-margin training of depen-
 11741 dency parsers. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
 11742 91–98.
- 11743 McDonald, R., K. Hannan, T. Neylon, M. Wells, and J. Reynar (2007). Structured models
 11744 for fine-to-coarse sentiment analysis. In *Proceedings of ACL*.
- 11745 McDonald, R. and F. Pereira (2006). Online learning of approximate dependency parsing
 11746 algorithms. In *Proceedings of the European Chapter of the Association for Computational
 11747 Linguistics (EACL)*.
- 11748 McKeown, K. (1992). *Text generation*. Cambridge University Press.
- 11749 McKeown, K., S. Rosenthal, K. Thadani, and C. Moore (2010). Time-efficient creation of
 11750 an accurate sentence fusion corpus. In *Proceedings of the North American Chapter of the
 11751 Association for Computational Linguistics (NAACL)*, pp. 317–320.
- 11752 McKeown, K. R., R. Barzilay, D. Evans, V. Hatzivassiloglou, J. L. Klavans, A. Nenkova,
 11753 C. Sable, B. Schiffman, and S. Sigelman (2002). Tracking and summarizing news on a
 11754 daily basis with columbia’s newsblaster. In *Proceedings of the second international confer-
 11755 ence on Human Language Technology Research*, pp. 280–285.
- 11756 McNamee, P. and H. T. Dang (2009). Overview of the tac 2009 knowledge base population
 11757 track. In *Text Analysis Conference (TAC)*, Volume 17, pp. 111–113.
- 11758 Medlock, B. and T. Briscoe (2007). Weakly supervised learning for hedge classification in
 11759 scientific literature. In *Proceedings of the Association for Computational Linguistics (ACL)*,
 11760 pp. 992–999.
- 11761 Mei, H., M. Bansal, and M. R. Walter (2016). What to talk about and how? selective gen-
 11762 eration using lstms with coarse-to-fine alignment. In *Proceedings of the North American
 11763 Chapter of the Association for Computational Linguistics (NAACL)*, pp. 720–730.
- 11764 Merity, S., N. S. Keskar, and R. Socher (2018). Regularizing and optimizing lstm language
 11765 models. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 11766 Merity, S., C. Xiong, J. Bradbury, and R. Socher (2017). Pointer sentinel mixture models.
 11767 In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 11768 Messud, C. (2014, June). A new ‘l’étranger’. *New York Review of Books*.

- 11769 Miao, Y. and P. Blunsom (2016). Language as a latent variable: Discrete generative mod-
11770 els for sentence compression. In *Proceedings of Empirical Methods for Natural Language*
11771 *Processing (EMNLP)*, pp. 319–328.
- 11772 Miao, Y., L. Yu, and P. Blunsom (2016). Neural variational inference for text processing. In
11773 *Proceedings of the International Conference on Machine Learning (ICML)*.
- 11774 Mihalcea, R., T. A. Chklovski, and A. Kilgarriff (2004, July). The senseval-3 english lexical
11775 sample task. In *Proceedings of SENSEVAL-3*, Barcelona, Spain, pp. 25–28. Association for
11776 Computational Linguistics.
- 11777 Mihalcea, R. and D. Radev (2011). *Graph-based natural language processing and information*
11778 *retrieval*. Cambridge University Press.
- 11779 Mikolov, T., K. Chen, G. Corrado, and J. Dean (2013). Efficient estimation of word repre-
11780 sentations in vector space. In *Proceedings of International Conference on Learning Represen-*
11781 *tations*.
- 11782 Mikolov, T., A. Deoras, D. Povey, L. Burget, and J. Cernocky (2011). Strategies for train-
11783 ing large scale neural network language models. In *Automatic Speech Recognition and*
11784 *Understanding (ASRU), 2011 IEEE Workshop on*, pp. 196–201. IEEE.
- 11785 Mikolov, T., M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur (2010). Recurrent
11786 neural network based language model. In *INTERSPEECH*, pp. 1045–1048.
- 11787 Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013). Distributed rep-
11788 resentations of words and phrases and their compositionality. In *Advances in Neural*
11789 *Information Processing Systems*, pp. 3111–3119.
- 11790 Mikolov, T., W.-t. Yih, and G. Zweig (2013). Linguistic regularities in continuous space
11791 word representations. In *Proceedings of the North American Chapter of the Association for*
11792 *Computational Linguistics (NAACL)*, pp. 746–751.
- 11793 Mikolov, T. and G. Zweig. Context dependent recurrent neural network language model.
11794 In *Proceedings of Spoken Language Technology (SLT)*, pp. 234–239.
- 11795 Miller, G. A., G. A. Heise, and W. Lichten (1951). The intelligibility of speech as a function
11796 of the context of the test materials. *Journal of experimental psychology* 41(5), 329.
- 11797 Miller, M., C. Sathi, D. Wiesenthal, J. Leskovec, and C. Potts (2011). Sentiment flow
11798 through hyperlink networks. In *Proceedings of the International Conference on Web and*
11799 *Social Media (ICWSM)*.
- 11800 Miller, S., J. Guinness, and A. Zamanian (2004). Name tagging with word clusters and
11801 discriminative training. In *Proceedings of the North American Chapter of the Association for*
11802 *Computational Linguistics (NAACL)*, pp. 337–342.

- 11803 Milne, D. and I. H. Witten (2008). Learning to link with wikipedia. In *Proceedings of the
11804 International Conference on Information and Knowledge Management (CIKM)*, pp. 509–518.
11805 ACM.
- 11806 Miltzakaki, E. and K. Kukich (2004). Evaluation of text coherence for electronic essay
11807 scoring systems. *Natural Language Engineering* 10(1), 25–55.
- 11808 Minka, T. P. (1999). From hidden markov models to linear dynamical systems. Tech. Rep.
11809 531, Vision and Modeling Group of Media Lab, MIT.
- 11810 Minsky, M. (1974). A framework for representing knowledge. Technical Report 306, MIT
11811 AI Laboratory.
- 11812 Minsky, M. and S. Papert (1969). *Perceptrons*. MIT press.
- 11813 Mintz, M., S. Bills, R. Snow, and D. Jurafsky (2009). Distant supervision for relation extrac-
11814 tion without labeled data. In *Proceedings of the Association for Computational Linguistics
11815 (ACL)*, pp. 1003–1011.
- 11816 Mirza, P., R. Sprugnoli, S. Tonelli, and M. Speranza (2014). Annotating causality in the
11817 tempeval-3 corpus. In *Proceedings of the EACL 2014 Workshop on Computational Ap-
11818 proaches to Causality in Language (CAtoCL)*, pp. 10–19.
- 11819 Misra, D. K. and Y. Artzi (2016). Neural shift-reduce ccg semantic parsing. In *Proceedings
11820 of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11821 Mitchell, J. and M. Lapata (2010). Composition in distributional models of semantics.
11822 *Cognitive Science* 34(8), 1388–1429.
- 11823 Miwa, M. and M. Bansal (2016). End-to-end relation extraction using lstms on sequences
11824 and tree structures. In *Proceedings of the Association for Computational Linguistics (ACL)*,
11825 pp. 1105–1116.
- 11826 Mnih, A. and G. Hinton (2007). Three new graphical models for statistical language mod-
11827 elling. In *Proceedings of the 24th international conference on Machine learning*, ICML '07,
11828 New York, NY, USA, pp. 641–648. ACM.
- 11829 Mnih, A. and G. E. Hinton (2008). A scalable hierarchical distributed language model. In
11830 *Neural Information Processing Systems (NIPS)*, pp. 1081–1088.
- 11831 Mnih, A. and Y. W. Teh (2012). A fast and simple algorithm for training neural probabilis-
11832 tic language models. In *Proceedings of the International Conference on Machine Learning
11833 (ICML)*.
- 11834 Mohammad, S. M. and P. D. Turney (2013). Crowdsourcing a word–emotion association
11835 lexicon. *Computational Intelligence* 29(3), 436–465.

- 11836 Mohri, M., F. Pereira, and M. Riley (2002). Weighted finite-state transducers in speech
11837 recognition. *Computer Speech & Language* 16(1), 69–88.
- 11838 Mohri, M., A. Rostamizadeh, and A. Talwalkar (2012). *Foundations of machine learning*.
11839 MIT press.
- 11840 Montague, R. (1973). The proper treatment of quantification in ordinary english. In *Ap-
11841 proaches to natural language*, pp. 221–242. Springer.
- 11842 Moore, J. D. and C. L. Paris (1993, dec). Planning text for advisory dialogues: Capturing
11843 intentional and rhetorical information. *Comput. Linguist.* 19(4), 651–694.
- 11844 Morante, R. and E. Blanco (2012). *sem 2012 shared task: Resolving the scope and fo-
11845 cus of negation. In *Proceedings of the First Joint Conference on Lexical and Computational
11846 Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2:
11847 Proceedings of the Sixth International Workshop on Semantic Evaluation*, pp. 265–274. Asso-
11848 ciation for Computational Linguistics.
- 11849 Morante, R. and W. Daelemans (2009). Learning the scope of hedge cues in biomedical
11850 texts. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language
11851 Processing*, pp. 28–36. Association for Computational Linguistics.
- 11852 Morante, R. and C. Sporleder (2012). Modality and negation: An introduction to the
11853 special issue. *Computational linguistics* 38(2), 223–260.
- 11854 Mostafazadeh, N., A. Grelish, N. Chambers, J. Allen, and L. Vanderwende (2016, June).
11855 Caters: Causal and temporal relation scheme for semantic annotation of event struc-
11856 tures. In *Proceedings of the Fourth Workshop on Events*, San Diego, California, pp. 51–61.
11857 Association for Computational Linguistics.
- 11858 Mueller, T., H. Schmid, and H. Schütze (2013). Efficient higher-order CRFs for morpholog-
11859 ical tagging. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
11860 pp. 322–332.
- 11861 Müller, C. and M. Strube (2006). Multi-level annotation of linguistic data with mmax2.
11862 *Corpus technology and language pedagogy: New resources, new tools, new methods* 3, 197–
11863 214.
- 11864 Muralidharan, A. and M. A. Hearst (2013). Supporting exploratory text analysis in litera-
11865 ture study. *Literary and linguistic computing* 28(2), 283–295.
- 11866 Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.
- 11867 Nakagawa, T., K. Inui, and S. Kurohashi (2010). Dependency tree-based sentiment classi-
11868 fication using crfs with hidden variables. In *Proceedings of the North American Chapter of
11869 the Association for Computational Linguistics (NAACL)*, pp. 786–794.

- 11870 Nakazawa, T., M. Yaguchi, K. Uchimoto, M. Utiyama, E. Sumita, S. Kurohashi, and H. Isahara (2016). ASPEC: Asian scientific paper excerpt corpus. In *Proceedings of the Language Resources and Evaluation Conference*, pp. 2204–2208.
- 11873 Navigli, R. (2009). Word sense disambiguation: A survey. *ACM Computing Surveys (CSUR)* 41(2), 10.
- 11875 Neal, R. M. and G. E. Hinton (1998). A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pp. 355–368. Springer.
- 11877 Nenkova, A. and K. McKeown (2012). A survey of text summarization techniques. In *Mining text data*, pp. 43–76. Springer.
- 11879 Neubig, G. (2017). Neural machine translation and sequence-to-sequence models: A tutorial. *arXiv preprint arXiv:1703.01619*.
- 11881 Neubig, G., C. Dyer, Y. Goldberg, A. Matthews, W. Ammar, A. Anastasopoulos, M. Balles-teros, D. Chiang, D. Clothiaux, T. Cohn, K. Duh, M. Faruqui, C. Gan, D. Garrette, Y. Ji, L. Kong, A. Kuncoro, G. Kumar, C. Malaviya, P. Michel, Y. Oda, M. Richardson, N. Saphra, S. Swayamdipta, and P. Yin (2017). Dynet: The dynamic neural network toolkit.
- 11886 Neubig, G., Y. Goldberg, and C. Dyer (2017). On-the-fly operation batching in dynamic computation graphs. In *Neural Information Processing Systems (NIPS)*.
- 11888 Neuhaus, P. and N. Bröker (1997). The complexity of recognition of linguistically adequate dependency grammars. In *eacl*, pp. 337–343.
- 11890 Newman, D., C. Chemudugunta, and P. Smyth (2006). Statistical entity-topic models. In *Proceedings of Knowledge Discovery and Data Mining (KDD)*, pp. 680–686.
- 11892 Ng, V. (2010). Supervised noun phrase coreference research: The first fifteen years. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pp. 1396–1411. Association for Computational Linguistics.
- 11895 Nguyen, D. and A. S. Dogruöz (2013). Word level language identification in online multi-lingual communication. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11898 Nguyen, D. T. and S. Joty (2017). A neural local coherence model. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1320–1330.
- 11900 Nigam, K., A. K. McCallum, S. Thrun, and T. Mitchell (2000). Text classification from labeled and unlabeled documents using em. *Machine learning* 39(2-3), 103–134.

- 11902 Nirenburg, S. and Y. Wilks (2001). What's in a symbol: ontology, representation and lan-
11903 guage. *Journal of Experimental & Theoretical Artificial Intelligence* 13(1), 9–23.
- 11904 Nivre, J. (2008). Algorithms for deterministic incremental dependency parsing. *Computa-*
11905 *tional Linguistics* 34(4), 513–553.
- 11906 Nivre, J., M.-C. de Marneffe, F. Ginter, Y. Goldberg, J. Hajič, C. D. Manning, R. McDonald,
11907 S. Petrov, S. Pyysalo, N. Silveira, R. Tsarfaty, and D. Zeman (2016, may). Universal de-
11908 pendencies v1: A multilingual treebank collection. In N. C. C. Chair), K. Choukri, T. De-
11909 clerck, S. Goggi, M. Grobelnik, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk,
11910 and S. Piperidis (Eds.), *Proceedings of the Tenth International Conference on Language Re-*
11911 *sources and Evaluation (LREC 2016)*, Paris, France. European Language Resources Asso-
11912 ciation (ELRA).
- 11913 Nivre, J. and J. Nilsson (2005). Pseudo-projective dependency parsing. In *Proceedings of the*
11914 *43rd Annual Meeting on Association for Computational Linguistics*, pp. 99–106. Association
11915 for Computational Linguistics.
- 11916 Novikoff, A. B. (1962). On convergence proofs on perceptrons. In *Proceedings of the Sym-*
11917 *posium on the Mathematical Theory of Automata*, Volume 12, pp. 615—622.
- 11918 Och, F. J. and H. Ney (2003). A systematic comparison of various statistical alignment
11919 models. *Computational linguistics* 29(1), 19–51.
- 11920 O'Connor, B., M. Krieger, and D. Ahn (2010). Tweetmotif: Exploratory search and topic
11921 summarization for twitter. In *Proceedings of the International Conference on Web and Social*
11922 *Media (ICWSM)*, pp. 384–385.
- 11923 Oflazer, K. and İ. Kuruöz (1994). Tagging and morphological disambiguation of turkish
11924 text. In *Proceedings of the fourth conference on Applied natural language processing*, pp. 144–
11925 149. Association for Computational Linguistics.
- 11926 Ohta, T., Y. Tateisi, and J.-D. Kim (2002). The genia corpus: An annotated research abstract
11927 corpus in molecular biology domain. In *Proceedings of the second international conference*
11928 *on Human Language Technology Research*, pp. 82–86. Morgan Kaufmann Publishers Inc.
- 11929 Onishi, T., H. Wang, M. Bansal, K. Gimpel, and D. McAllester (2016). Who did what: A
11930 large-scale person-centered cloze dataset. In *Proceedings of Empirical Methods for Natural*
11931 *Language Processing (EMNLP)*, pp. 2230–2235.
- 11932 Owoputi, O., B. O'Connor, C. Dyer, K. Gimpel, N. Schneider, and N. A. Smith (2013).
11933 Improved part-of-speech tagging for online conversational text with word clusters. In
11934 *Proceedings of the North American Chapter of the Association for Computational Linguistics*
11935 (*NAACL*), pp. 380–390.

- 11936 Packard, W., E. M. Bender, J. Read, S. Oepen, and R. Dridan (2014). Simple negation
 11937 scope resolution through deep parsing: A semantic solution to a semantic problem. In
 11938 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 69–78.
- 11939 Paice, C. D. (1990). Another stemmer. In *ACM SIGIR Forum*, Volume 24, pp. 56–61.
- 11940 Pak, A. and P. Paroubek (2010). Twitter as a corpus for sentiment analysis and opinion
 11941 mining. In *LREC*, Volume 10, pp. 1320–1326.
- 11942 Palmer, F. R. (2001). *Mood and modality*. Cambridge University Press.
- 11943 Palmer, M., D. Gildea, and P. Kingsbury (2005). The proposition bank: An annotated
 11944 corpus of semantic roles. *Computational linguistics* 31(1), 71–106.
- 11945 Pan, S. J. and Q. Yang (2010). A survey on transfer learning. *IEEE Transactions on knowledge
 11946 and data engineering* 22(10), 1345–1359.
- 11947 Pan, X., T. Cassidy, U. Hermjakob, H. Ji, and K. Knight (2015). Unsupervised entity linking
 11948 with abstract meaning representation. In *Proceedings of the North American Chapter of the
 11949 Association for Computational Linguistics (NAACL)*, pp. 1130–1139.
- 11950 Pang, B. and L. Lee (2004). A sentimental education: Sentiment analysis using subjectivity
 11951 summarization based on minimum cuts. In *Proceedings of the Association for Compu-
 11952 tational Linguistics (ACL)*, pp. 271–278.
- 11953 Pang, B. and L. Lee (2005). Seeing stars: Exploiting class relationships for sentiment cate-
 11954 gorization with respect to rating scales. In *Proceedings of the Association for Computational
 11955 Linguistics (ACL)*, pp. 115–124.
- 11956 Pang, B. and L. Lee (2008). Opinion mining and sentiment analysis. *Foundations and trends
 11957 in information retrieval* 2(1–2), 1–135.
- 11958 Pang, B., L. Lee, and S. Vaithyanathan (2002). Thumbs up?: sentiment classification using
 11959 machine learning techniques. In *Proceedings of Empirical Methods for Natural Language
 11960 Processing (EMNLP)*, pp. 79–86.
- 11961 Papineni, K., S. Roukos, T. Ward, and W.-J. Zhu (2002). Bleu: a method for automatic
 11962 evaluation of machine translation. In *Proceedings of the Association for Computational
 11963 Linguistics (ACL)*, pp. 311–318.
- 11964 Park, J. and C. Cardie (2012). Improving implicit discourse relation recognition through
 11965 feature set optimization. In *Proceedings of the Special Interest Group on Discourse and Dia-
 11966 logue (SIGDIAL)*, pp. 108–112.
- 11967 Parsons, T. (1990). *Events in the Semantics of English*, Volume 5. MIT Press.

- 11968 Pascanu, R., T. Mikolov, and Y. Bengio (2013). On the difficulty of training recurrent neural
11969 networks. In *Proceedings of the 30th International Conference on Machine Learning (ICML-*
11970 *13)*, pp. 1310–1318.
- 11971 Paul, M., M. Federico, and S. Stüker (2010). Overview of the iwslt 2010 evaluation cam-
11972 paign. In *International Workshop on Spoken Language Translation (IWSLT) 2010*.
- 11973 Pedersen, T., S. Patwardhan, and J. Michelizzi (2004). Wordnet::similarity - measuring the
11974 relatedness of concepts. In *Proceedings of the North American Chapter of the Association for*
11975 *Computational Linguistics (NAACL)*, pp. 38–41.
- 11976 Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blon-
11977 del, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau,
11978 M. Brucher, M. Perrot, and E. Duchesnay (2011). Scikit-learn: Machine learning in
11979 Python. *Journal of Machine Learning Research* 12, 2825–2830.
- 11980 Pei, W., T. Ge, and B. Chang (2015). An effective neural network model for graph-based
11981 dependency parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*,
11982 pp. 313–322.
- 11983 Peldszus, A. and M. Stede (2013). From argument diagrams to argumentation mining
11984 in texts: A survey. *International Journal of Cognitive Informatics and Natural Intelligence*
11985 (*IJCINI*) 7(1), 1–31.
- 11986 Peldszus, A. and M. Stede (2015). An annotated corpus of argumentative microtexts. In
11987 *Proceedings of the First Conference on Argumentation*.
- 11988 Peng, F., F. Feng, and A. McCallum (2004). Chinese segmentation and new word detec-
11989 tion using conditional random fields. In *Proceedings of the International Conference on*
11990 *Computational Linguistics (COLING)*, pp. 562.
- 11991 Pennington, J., R. Socher, and C. Manning (2014). Glove: Global vectors for word repre-
11992 sentation. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
11993 pp. 1532–1543.
- 11994 Pereira, F. and Y. Schabes (1992). Inside-outside reestimation from partially bracketed
11995 corpora. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 128–
11996 135.
- 11997 Pereira, F. C. N. and S. M. Shieber (2002). *Prolog and natural-language analysis*. Microtome
11998 Publishing.
- 11999 Peters, M. E., M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer
12000 (2018). Deep contextualized word representations. In *Proceedings of the North American*
12001 *Chapter of the Association for Computational Linguistics (NAACL)*.

- 12002 Peterson, W. W., T. G. Birdsall, and W. C. Fox (1954). The theory of signal detectability.
12003 *Transactions of the IRE professional group on information theory* 4(4), 171–212.
- 12004 Petrov, S., L. Barrett, R. Thibaux, and D. Klein (2006). Learning accurate, compact, and in-
12005 terpretable tree annotation. In *Proceedings of the Association for Computational Linguistics*
12006 (*ACL*).
- 12007 Petrov, S., D. Das, and R. McDonald (2012, May). A universal part-of-speech tagset. In
12008 *Proceedings of LREC*.
- 12009 Petrov, S. and R. McDonald (2012). Overview of the 2012 shared task on parsing the web.
12010 In *Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL)*,
12011 Volume 59.
- 12012 Pinker, S. (2003). *The language instinct: How the mind creates language*. Penguin UK.
- 12013 Pinter, Y., R. Guthrie, and J. Eisenstein (2017). Mimicking word embeddings using
12014 subword RNNs. In *Proceedings of Empirical Methods for Natural Language Processing*
12015 (*EMNLP*).
- 12016 Pitler, E., A. Louis, and A. Nenkova (2009). Automatic sense prediction for implicit dis-
12017 course relations in text. In *Proceedings of the Association for Computational Linguistics*
12018 (*ACL*).
- 12019 Pitler, E. and A. Nenkova (2009). Using syntax to disambiguate explicit discourse con-
12020 nectives in text. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
12021 13–16.
- 12022 Pitler, E., M. Raghupathy, H. Mehta, A. Nenkova, A. Lee, and A. Joshi (2008). Easily iden-
12023 tifiable discourse relations. In *Proceedings of the International Conference on Computational*
12024 *Linguistics (COLING)*, pp. 87–90.
- 12025 Plank, B., A. Søgaard, and Y. Goldberg (2016). Multilingual part-of-speech tagging with
12026 bidirectional long short-term memory models and auxiliary loss. In *Proceedings of the*
12027 *Association for Computational Linguistics (ACL)*.
- 12028 Poesio, M., R. Stevenson, B. Di Eugenio, and J. Hitzeman (2004). Centering: A parametric
12029 theory and its instantiations. *Computational linguistics* 30(3), 309–363.
- 12030 Polanyi, L. and A. Zaenen (2006). Contextual valence shifters. In *Computing attitude and*
12031 *affect in text: Theory and applications*. Springer.
- 12032 Ponzetto, S. P. and M. Strube (2006). Exploiting semantic role labeling, wordnet and
12033 wikipedia for coreference resolution. In *Proceedings of the North American Chapter of*
12034 *the Association for Computational Linguistics (NAACL)*, pp. 192–199.

- 12035 Ponzetto, S. P. and M. Strube (2007). Knowledge derived from wikipedia for computing
12036 semantic relatedness. *Journal of Artificial Intelligence Research* 30, 181–212.
- 12037 Poon, H. and P. Domingos (2008). Joint unsupervised coreference resolution with markov
12038 logic. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.
12039 650–659.
- 12040 Poon, H. and P. Domingos (2009). Unsupervised semantic parsing. In *Proceedings of Em-
12041 pirical Methods for Natural Language Processing (EMNLP)*, pp. 1–10.
- 12042 Popel, M., D. Marecek, J. Stepánek, D. Zeman, and Z. Zabokrtský (2013). Coordination
12043 structures in dependency treebanks. In *Proceedings of the Association for Computational
12044 Linguistics (ACL)*, pp. 517–527.
- 12045 Popescu, A.-M., O. Etzioni, and H. Kautz (2003). Towards a theory of natural language
12046 interfaces to databases. In *Proceedings of Intelligent User Interfaces (IUI)*, pp. 149–157.
- 12047 Poplack, S. (1980). Sometimes i'll start a sentence in spanish y termino en español: toward
12048 a typology of code-switching1. *Linguistics* 18(7-8), 581–618.
- 12049 Porter, M. F. (1980). An algorithm for suffix stripping. *Program* 14(3), 130–137.
- 12050 Prabhakaran, V., O. Rambow, and M. Diab (2010). Automatic committed belief tagging.
12051 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.
12052 1014–1022.
- 12053 Pradhan, S., X. Luo, M. Recasens, E. Hovy, V. Ng, and M. Strube (2014). Scoring corefer-
12054 ence partitions of predicted mentions: A reference implementation. In *Proceedings of the
12055 Association for Computational Linguistics (ACL)*, pp. 30–35.
- 12056 Pradhan, S., L. Ramshaw, M. Marcus, M. Palmer, R. Weischedel, and N. Xue (2011).
12057 CoNLL-2011 shared task: Modeling unrestricted coreference in OntoNotes. In *Proceed-
12058 ings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*,
12059 pp. 1–27. Association for Computational Linguistics.
- 12060 Pradhan, S., W. Ward, K. Hacioglu, J. H. Martin, and D. Jurafsky (2005). Semantic role
12061 labeling using different syntactic views. In *Proceedings of the Association for Computational
12062 Linguistics (ACL)*, pp. 581–588.
- 12063 Prasad, R., N. Dinesh, A. Lee, E. Miltsakaki, L. Robaldo, A. Joshi, and B. Webber (2008).
12064 The Penn Discourse Treebank 2.0. In *Proceedings of LREC*.
- 12065 Punyakanok, V., D. Roth, and W.-t. Yih (2008). The importance of syntactic parsing and
12066 inference in semantic role labeling. *Computational Linguistics* 34(2), 257–287.

- 12067 Pustejovsky, J., P. Hanks, R. Sauri, A. See, R. Gaizauskas, A. Setzer, D. Radev, B. Sundheim,
12068 D. Day, L. Ferro, et al. (2003). The timebank corpus. In *Corpus linguistics*, Volume 2003,
12069 pp. 40. Lancaster, UK.
- 12070 Pustejovsky, J., B. Ingria, R. Sauri, J. Castano, J. Littman, R. Gaizauskas, A. Setzer, G. Katz,
12071 and I. Mani (2005). The specification language timeml. In *The language of time: A reader*,
12072 pp. 545–557. Oxford University Press.
- 12073 Qin, L., Z. Zhang, H. Zhao, Z. Hu, and E. Xing (2017). Adversarial connective-exploiting
12074 networks for implicit discourse relation classification. In *Proceedings of the Association
12075 for Computational Linguistics (ACL)*, pp. 1006–1017.
- 12076 Qiu, G., B. Liu, J. Bu, and C. Chen (2011). Opinion word expansion and target extraction
12077 through double propagation. *Computational linguistics* 37(1), 9–27.
- 12078 Quattoni, A., S. Wang, L.-P. Morency, M. Collins, and T. Darrell (2007). Hidden conditional
12079 random fields. *IEEE transactions on pattern analysis and machine intelligence* 29(10).
- 12080 Rahman, A. and V. Ng (2011). Narrowing the modeling gap: a cluster-ranking approach
12081 to coreference resolution. *Journal of Artificial Intelligence Research* 40, 469–521.
- 12082 Rajpurkar, P., J. Zhang, K. Lopyrev, and P. Liang (2016). Squad: 100,000+ questions for
12083 machine comprehension of text. In *Proceedings of Empirical Methods for Natural Language
12084 Processing (EMNLP)*, pp. 2383–2392.
- 12085 Ranzato, M., S. Chopra, M. Auli, and W. Zaremba (2016). Sequence level training with
12086 recurrent neural networks. In *Proceedings of the International Conference on Learning Rep-
12087 resentations (ICLR)*.
- 12088 Rao, D., D. Yarowsky, A. Shreevats, and M. Gupta (2010). Classifying latent user attributes
12089 in twitter. In *Proceedings of Workshop on Search and mining user-generated contents*.
- 12090 Ratinov, L. and D. Roth (2009). Design challenges and misconceptions in named entity
12091 recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language
12092 Learning*, pp. 147–155. Association for Computational Linguistics.
- 12093 Ratinov, L., D. Roth, D. Downey, and M. Anderson (2011). Local and global algorithms
12094 for disambiguation to wikipedia. In *Proceedings of the Association for Computational Lin-
12095 guistics (ACL)*, pp. 1375–1384.
- 12096 Ratliff, N. D., J. A. Bagnell, and M. Zinkevich (2007). (approximate) subgradient methods
12097 for structured prediction. In *Proceedings of Artificial Intelligence and Statistics (AISTATS)*,
12098 pp. 380–387.

- 12099 Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In *emnlp*,
12100 pp. 133–142.
- 12101 Ratnaparkhi, A., J. Reynar, and S. Roukos (1994). A maximum entropy model for preposi-
12102 tional phrase attachment. In *Proceedings of the workshop on Human Language Technology*,
12103 pp. 250–255. Association for Computational Linguistics.
- 12104 Read, J. (2005). Using emoticons to reduce dependency in machine learning techniques for
12105 sentiment classification. In *Proceedings of the ACL student research workshop*, pp. 43–48.
12106 Association for Computational Linguistics.
- 12107 Reisinger, D., R. Rudinger, F. Ferraro, C. Harman, K. Rawlins, and B. V. Durme (2015).
12108 Semantic proto-roles. *Transactions of the Association for Computational Linguistics* 3, 475–
12109 488.
- 12110 Reisinger, J. and R. J. Mooney (2010). Multi-prototype vector-space models of word mean-
12111 ing. In *Proceedings of the North American Chapter of the Association for Computational Lin-*
12112 *guistics (NAACL)*, pp. 109–117.
- 12113 Reiter, E. and R. Dale (2000). *Building natural language generation systems*. Cambridge
12114 university press.
- 12115 Resnik, P., M. B. Olsen, and M. Diab (1999). The bible as a parallel corpus: Annotating the
12116 ‘book of 2000 tongues’. *Computers and the Humanities* 33(1-2), 129–153.
- 12117 Resnik, P. and N. A. Smith (2003). The web as a parallel corpus. *Computational Linguis-*
12118 *tics* 29(3), 349–380.
- 12119 Ribeiro, F. N., M. Araújo, P. Gonçalves, M. A. Gonçalves, and F. Benevenuto (2016).
12120 Sentibench-a benchmark comparison of state-of-the-practice sentiment analysis meth-
12121 ods. *EPJ Data Science* 5(1), 1–29.
- 12122 Richardson, M., C. J. Burges, and E. Renshaw (2013). MCTest: A challenge dataset for
12123 the open-domain machine comprehension of text. In *Proceedings of Empirical Methods for*
12124 *Natural Language Processing (EMNLP)*, pp. 193–203.
- 12125 Riedel, S., L. Yao, and A. McCallum (2010). Modeling relations and their mentions without
12126 labeled text. In *Proceedings of the European Conference on Machine Learning and Principles*
12127 *and Practice of Knowledge Discovery in Databases (ECML)*, pp. 148–163.
- 12128 Riedl, M. O. and R. M. Young (2010). Narrative planning: Balancing plot and character.
12129 *Journal of Artificial Intelligence Research* 39, 217–268.
- 12130 Rieser, V. and O. Lemon (2011). *Reinforcement learning for adaptive dialogue systems: a data-*
12131 *driven methodology for dialogue management and natural language generation*. Springer Sci-
12132 ence & Business Media.

- 12133 Riloff, E. (1996). Automatically generating extraction patterns from untagged text. In
 12134 *Proceedings of the national conference on artificial intelligence*, pp. 1044–1049.
- 12135 Riloff, E. and J. Wiebe (2003). Learning extraction patterns for subjective expressions. In
 12136 *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pp.
 12137 105–112. Association for Computational Linguistics.
- 12138 Ritchie, G. (2001). Current directions in computational humour. *Artificial Intelligence Re-*
 12139 *view* 16(2), 119–135.
- 12140 Ritter, A., C. Cherry, and W. B. Dolan (2011). Data-driven response generation in social
 12141 media. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.
 12142 583–593.
- 12143 Roark, B., M. Saraclar, and M. Collins (2007). Discriminative i_l / n_l / i_l -gram language
 12144 modeling. *Computer Speech & Language* 21(2), 373–392.
- 12145 Robert, C. and G. Casella (2013). *Monte Carlo statistical methods*. Springer Science & Busi-
 12146 ness Media.
- 12147 Rosenfeld, R. (1996). A maximum entropy approach to adaptive statistical language mod-
 12148 elling. *Computer Speech & Language* 10(3), 187–228.
- 12149 Ross, S., G. Gordon, and D. Bagnell (2011). A reduction of imitation learning and struc-
 12150 tured prediction to no-regret online learning. In *Proceedings of Artificial Intelligence and*
 12151 *Statistics (AISTATS)*, pp. 627–635.
- 12152 Roy, N., J. Pineau, and S. Thrun (2000). Spoken dialogue management using probabilistic
 12153 reasoning. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 93–
 12154 100.
- 12155 Rudnicky, A. and W. Xu (1999). An agenda-based dialog management architecture for
 12156 spoken language systems. In *IEEE Automatic Speech Recognition and Understanding Work-
 12157 shop*, Volume 13.
- 12158 Rush, A. M., S. Chopra, and J. Weston (2015). A neural attention model for abstractive sen-
 12159 tence summarization. In *Proceedings of Empirical Methods for Natural Language Processing*
 12160 (*EMNLP*), pp. 379–389.
- 12161 Rush, A. M., D. Sontag, M. Collins, and T. Jaakkola (2010). On dual decomposition and
 12162 linear programming relaxations for natural language processing. In *Proceedings of Em-
 12163 pirical Methods for Natural Language Processing (EMNLP)*, pp. 1–11.
- 12164 Russell, S. J. and P. Norvig (2009). *Artificial intelligence: a modern approach* (3rd ed.). Prentice
 12165 Hall.

- 12166 Rutherford, A., V. Demberg, and N. Xue (2017). A systematic study of neural discourse
12167 models for implicit discourse relation. In *Proceedings of the European Chapter of the Asso-*
12168 *ciation for Computational Linguistics (EACL)*, pp. 281–291.
- 12169 Rutherford, A. T. and N. Xue (2014). Discovering implicit discourse relations through
12170 brown cluster pair representation and coreference patterns. In *Proceedings of the Euro-*
12171 *pean Chapter of the Association for Computational Linguistics (EACL)*.
- 12172 Sag, I. A., T. Baldwin, F. Bond, A. Copestake, and D. Flickinger (2002). Multiword expres-
12173 sions: A pain in the neck for nlp. In *International Conference on Intelligent Text Processing*
12174 and *Computational Linguistics*, pp. 1–15. Springer.
- 12175 Sagae, K. (2009). Analysis of discourse structure with syntactic dependencies and data-
12176 driven shift-reduce parsing. In *Proceedings of the 11th International Conference on Parsing*
12177 *Technologies*, pp. 81–84.
- 12178 Santos, C. D. and B. Zadrozny (2014). Learning character-level representations for part-of-
12179 speech tagging. In *Proceedings of the International Conference on Machine Learning (ICML)*,
12180 pp. 1818–1826.
- 12181 Sato, M.-A. and S. Ishii (2000). On-line em algorithm for the normalized gaussian network.
12182 *Neural computation* 12(2), 407–432.
- 12183 Saurí, R. and J. Pustejovsky (2009). Factbank: a corpus annotated with event factuality.
12184 *Language resources and evaluation* 43(3), 227.
- 12185 Saxe, A. M., J. L. McClelland, and S. Ganguli (2014). Exact solutions to the nonlinear
12186 dynamics of learning in deep linear neural networks. In *Proceedings of the International*
12187 *Conference on Learning Representations (ICLR)*.
- 12188 Schank, R. C. and R. Abelson (1977). *Scripts, goals, plans, and understanding*. Hillsdale, NJ:
12189 Erlbaum.
- 12190 Schapire, R. E. and Y. Singer (2000). Boostexter: A boosting-based system for text catego-
12191 *Machine learning* 39(2-3), 135–168.
- 12192 Schaul, T., S. Zhang, and Y. LeCun (2013). No more pesky learning rates. In *Proceedings of*
12193 *the International Conference on Machine Learning (ICML)*, pp. 343–351.
- 12194 Schnabel, T., I. Labutov, D. Mimno, and T. Joachims (2015). Evaluation methods for un-
12195 supervised word embeddings. In *Proceedings of Empirical Methods for Natural Language*
12196 *Processing (EMNLP)*, pp. 298–307.
- 12197 Schneider, N., J. Flanigan, and T. O’Gorman (2015). The logic of amr: Practical, unified,
12198 graph-based sentence semantics for nlp. In *Proceedings of the North American Chapter of*
12199 *the Association for Computational Linguistics (NAACL)*, pp. 4–5.

- 12200 Schütze, H. (1998). Automatic word sense discrimination. *Computational linguistics* 24(1),
12201 97–123.
- 12202 Schwarm, S. E. and M. Ostendorf (2005). Reading level assessment using support vector
12203 machines and statistical language models. In *Proceedings of the Association for Compu-*
12204 *tational Linguistics (ACL)*, pp. 523–530.
- 12205 See, A., P. J. Liu, and C. D. Manning (2017). Get to the point: Summarization with pointer-
12206 generator networks. In *Proceedings of the Association for Computational Linguistics (ACL)*,
12207 pp. 1073–1083.
- 12208 Sennrich, R., B. Haddow, and A. Birch (2016). Neural machine translation of rare words
12209 with subword units. In *Proceedings of the Association for Computational Linguistics (ACL)*,
12210 pp. 1715–1725.
- 12211 Serban, I. V., A. Sordoni, Y. Bengio, A. C. Courville, and J. Pineau (2016). Building end-to-
12212 end dialogue systems using generative hierarchical neural network models. In *Proceed-*
12213 *ings of the National Conference on Artificial Intelligence (AAAI)*, pp. 3776–3784.
- 12214 Settles, B. (2012). Active learning. *Synthesis Lectures on Artificial Intelligence and Machine*
12215 *Learning* 6(1), 1–114.
- 12216 Shang, L., Z. Lu, and H. Li (2015). Neural responding machine for short-text conversation.
12217 In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1577–1586.
- 12218 Shen, D. and M. Lapata (2007). Using semantic roles to improve question answering. In
12219 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 12–21.
- 12220 Shen, S., Y. Cheng, Z. He, W. He, H. Wu, M. Sun, and Y. Liu (2016). Minimum risk train-
12221 ing for neural machine translation. In *Proceedings of the Association for Computation-*
12222 *al Linguistics (ACL)*, pp. 1683–1692.
- 12223 Shen, W., J. Wang, and J. Han (2015). Entity linking with a knowledge base: Issues, tech-
12224 niques, and solutions. *IEEE Transactions on Knowledge and Data Engineering* 27(2), 443–
12225 460.
- 12226 Shieber, S. M. (1985). Evidence against the context-freeness of natural language. *Linguistics*
12227 and Philosophy
- 12228 Siegelmann, H. T. and E. D. Sontag (1995). On the computational power of neural nets.
12229 *Journal of computer and system sciences* 50(1), 132–150.
- 12230 Singh, S., A. Subramanya, F. Pereira, and A. McCallum (2011). Large-scale cross-
12231 document coreference using distributed inference and hierarchical models. In *Proceed-*
12232 *ings of the Association for Computational Linguistics (ACL)*, pp. 793–803.

- 12233 Sipser, M. (2012). *Introduction to the Theory of Computation*. Cengage Learning.
- 12234 Smith, D. A. and J. Eisner (2006). Minimum risk annealing for training log-linear models.
12235 In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 787–794.
- 12236 Smith, D. A. and J. Eisner (2008). Dependency parsing by belief propagation. In *Proceed-
12237 ings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 145–156.
- 12238 Smith, D. A. and N. A. Smith (2007). Probabilistic models of nonprojective dependency
12239 trees. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.
12240 132–140.
- 12241 Smith, N. A. (2011). Linguistic structure prediction. *Synthesis Lectures on Human Language
12242 Technologies* 4(2), 1–274.
- 12243 Snover, M., B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul (2006). A study of transla-
12244 tion edit rate with targeted human annotation. In *Proceedings of association for machine
12245 translation in the Americas*, Volume 200.
- 12246 Snow, R., B. O’Connor, D. Jurafsky, and A. Y. Ng (2008). Cheap and fast—but is it good?:
12247 evaluating non-expert annotations for natural language tasks. In *Proceedings of Empirical
12248 Methods for Natural Language Processing (EMNLP)*, pp. 254–263.
- 12249 Snyder, B. and R. Barzilay (2007). Database-text alignment via structured multilabel classi-
12250 fication. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*,
12251 pp. 1713–1718.
- 12252 Socher, R., J. Bauer, C. D. Manning, and A. Y. Ng (2013). Parsing with compositional vector
12253 grammars. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 12254 Socher, R., B. Huval, C. D. Manning, and A. Y. Ng (2012). Semantic compositionality
12255 through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Em-
12256 pirical Methods in Natural Language Processing and Computational Natural Language Learn-
12257 ing*, pp. 1201–1211. Association for Computational Linguistics.
- 12258 Socher, R., A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts (2013).
12259 Recursive deep models for semantic compositionality over a sentiment treebank. In
12260 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12261 Søgaard, A. (2013). Semi-supervised learning and domain adaptation in natural language
12262 processing. *Synthesis Lectures on Human Language Technologies* 6(2), 1–103.
- 12263 Solorio, T. and Y. Liu (2008). Learning to predict code-switching points. In *Proceedings
12264 of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 973–981. Association
12265 for Computational Linguistics.

- 12266 Somasundaran, S., G. Namata, J. Wiebe, and L. Getoor (2009). Supervised and unsupervised methods in employing discourse relations for improving opinion polarity classification. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12269 Somasundaran, S. and J. Wiebe (2009). Recognizing stances in online debates. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 226–234.
- 12271 Song, L., B. Boots, S. M. Siddiqi, G. J. Gordon, and A. J. Smola (2010). Hilbert space embeddings of hidden markov models. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 991–998.
- 12274 Song, L., Y. Zhang, X. Peng, Z. Wang, and D. Gildea (2016). Amr-to-text generation as a traveling salesman problem. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 2084–2089.
- 12277 Soon, W. M., H. T. Ng, and D. C. Y. Lim (2001). A machine learning approach to coreference resolution of noun phrases. *Computational linguistics* 27(4), 521–544.
- 12279 Sordoni, A., M. Galley, M. Auli, C. Brockett, Y. Ji, M. Mitchell, J.-Y. Nie, J. Gao, and B. Dolan (2015). A neural network approach to context-sensitive generation of conversational responses. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.
- 12283 Soriceut, R. and D. Marcu (2003). Sentence level discourse parsing using syntactic and lexical information. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 149–156.
- 12286 Sowa, J. F. (2000). *Knowledge representation: logical, philosophical, and computational foundations*. Pacific Grove, CA: Brooks/Cole.
- 12288 Spärck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation* 28(1), 11–21.
- 12290 Spitkovsky, V. I., H. Alshawi, D. Jurafsky, and C. D. Manning (2010). Viterbi training improves unsupervised dependency parsing. In *CONLL*, pp. 9–17.
- 12292 Sporleder, C. and M. Lapata (2005). Discourse chunking and its application to sentence compression. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 257–264.
- 12295 Sproat, R., A. Black, S. Chen, S. Kumar, M. Ostendorf, and C. Richards (2001). Normalization of non-standard words. *Computer Speech & Language* 15(3), 287–333.
- 12297 Sproat, R., W. Gale, C. Shih, and N. Chang (1996). A stochastic finite-state word-segmentation algorithm for chinese. *Computational linguistics* 22(3), 377–404.

- 12299 Sra, S., S. Nowozin, and S. J. Wright (2012). *Optimization for machine learning*. MIT Press.
- 12300 Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014).
- 12301 Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1), 1929–1958.
- 12302
- 12303 Srivastava, R. K., K. Greff, and J. Schmidhuber (2015). Training very deep networks. In
- 12304 *Neural Information Processing Systems (NIPS)*, pp. 2377–2385.
- 12305 Stab, C. and I. Gurevych (2014a). Annotating argument components and relations in per-
- 12306 suasive essays. In *Proceedings of the International Conference on Computational Linguistics*
- 12307 (COLING), pp. 1501–1510.
- 12308 Stab, C. and I. Gurevych (2014b). Identifying argumentative discourse structures in per-
- 12309 suasive essays. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Lan-*
- 12310 *guage Processing (EMNLP)*, pp. 46–56.
- 12311 Stede, M. (2011, nov). *Discourse Processing*, Volume 4 of *Synthesis Lectures on Human Lan-*
- 12312 *guage Technologies*. Morgan & Claypool Publishers.
- 12313 Steedman, M. and J. Baldridge (2011). Combinatory categorial grammar. In *Non-*
- 12314 *Transformational Syntax: Formal and Explicit Models of Grammar*. Wiley-Blackwell.
- 12315 Stenetorp, P., S. Pyysalo, G. Topić, T. Ohta, S. Ananiadou, and J. Tsujii (2012). Brat: a web-
- 12316 based tool for nlp-assisted text annotation. In *Proceedings of the European Chapter of the*
- 12317 *Association for Computational Linguistics (EACL)*, pp. 102–107.
- 12318 Stern, M., J. Andreas, and D. Klein (2017). A minimal span-based neural constituency
- 12319 parser. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 12320 Stolcke, A., K. Ries, N. Coccaro, E. Shriberg, R. Bates, D. Jurafsky, P. Taylor, R. Martin,
- 12321 C. Van Ess-Dykema, and M. Meteer (2000). Dialogue act modeling for automatic tag-
- 12322 ging and recognition of conversational speech. *Computational linguistics* 26(3), 339–373.
- 12323 Stone, P. J. (1966). *The General Inquirer: A Computer Approach to Content Analysis*. The MIT
- 12324 Press.
- 12325 Stoyanov, V., N. Gilbert, C. Cardie, and E. Riloff (2009). Conundrums in noun phrase
- 12326 coreference resolution: Making sense of the state-of-the-art. In *Proceedings of the Associa-*
- 12327 *tion for Computational Linguistics (ACL)*, pp. 656–664.
- 12328 Strang, G. (2016). *Introduction to linear algebra* (Fifth ed.). Wellesley, MA: Wellesley-
- 12329 Cambridge Press.

- 12330 Strubell, E., P. Verga, D. Belanger, and A. McCallum (2017). Fast and accurate entity recognition
 12331 with iterated dilated convolutions. In *Proceedings of Empirical Methods for Natural
 12332 Language Processing (EMNLP)*.
- 12333 Suchanek, F. M., G. Kasneci, and G. Weikum (2007). Yago: a core of semantic knowledge.
 12334 In *Proceedings of the Conference on World-Wide Web (WWW)*, pp. 697–706.
- 12335 Sun, X., T. Matsuzaki, D. Okanohara, and J. Tsujii (2009). Latent variable perceptron algo-
 12336 rithm for structured classification. In *Proceedings of the International Joint Conference on
 12337 Artificial Intelligence (IJCAI)*, Volume 9, pp. 1236–1242.
- 12338 Sun, Y., L. Lin, D. Tang, N. Yang, Z. Ji, and X. Wang (2015). Modeling mention, context
 12339 and entity with neural networks for entity disambiguation. In *IJCAI*, pp. 1333–1339.
- 12340 Sundermeyer, M., R. Schlüter, and H. Ney (2012). Lstm neural networks for language
 12341 modeling. In *INTERSPEECH*.
- 12342 Surdeanu, M., J. Tibshirani, R. Nallapati, and C. D. Manning (2012). Multi-instance multi-
 12343 label learning for relation extraction. In *Proceedings of Empirical Methods for Natural Lan-
 12344 guage Processing (EMNLP)*, pp. 455–465.
- 12345 Sutskever, I., O. Vinyals, and Q. V. Le (2014). Sequence to sequence learning with neural
 12346 networks. In *Neural Information Processing Systems (NIPS)*, pp. 3104–3112.
- 12347 Sutton, R. S. and A. G. Barto (1998). *Reinforcement learning: An introduction*, Volume 1. MIT
 12348 press Cambridge.
- 12349 Sutton, R. S., D. A. McAllester, S. P. Singh, and Y. Mansour (2000). Policy gradient methods
 12350 for reinforcement learning with function approximation. In *Neural Information Process-
 12351 ing Systems (NIPS)*, pp. 1057–1063.
- 12352 Taboada, M., J. Brooke, M. Tofiloski, K. Voll, and M. Stede (2011). Lexicon-based methods
 12353 for sentiment analysis. *Computational linguistics* 37(2), 267–307.
- 12354 Taboada, M. and W. C. Mann (2006). Rhetorical structure theory: Looking back and mov-
 12355 ing ahead. *Discourse studies* 8(3), 423–459.
- 12356 Täckström, O., K. Ganchev, and D. Das (2015). Efficient inference and structured learning
 12357 for semantic role labeling. *Transactions of the Association for Computational Linguistics* 3,
 12358 29–41.
- 12359 Täckström, O., R. McDonald, and J. Uszkoreit (2012). Cross-lingual word clusters for
 12360 direct transfer of linguistic structure. In *Proceedings of the North American Chapter of the
 12361 Association for Computational Linguistics (NAACL)*, pp. 477–487.

- 12362 Tang, D., B. Qin, and T. Liu (2015). Document modeling with gated recurrent neural net-
12363 work for sentiment classification. In *Proceedings of Empirical Methods for Natural Language
12364 Processing (EMNLP)*, pp. 1422–1432.
- 12365 Taskar, B., C. Guestrin, and D. Koller (2003). Max-margin markov networks. In *Neural
12366 Information Processing Systems (NIPS)*.
- 12367 Tausczik, Y. R. and J. W. Pennebaker (2010). The psychological meaning of words: LIWC
12368 and computerized text analysis methods. *Journal of Language and Social Psychology* 29(1),
12369 24–54.
- 12370 Teh, Y. W. (2006). A hierarchical bayesian language model based on pitman-yor processes.
12371 In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 985–992.
- 12372 Tesnière, L. (1966). *Éléments de syntaxe structurale* (second ed.). Paris: Klincksieck.
- 12373 Teufel, S., J. Carletta, and M. Moens (1999). An annotation scheme for discourse-level
12374 argumentation in research articles. In *Proceedings of the European Chapter of the Association
12375 for Computational Linguistics (EACL)*, pp. 110–117.
- 12376 Teufel, S. and M. Moens (2002). Summarizing scientific articles: experiments with rele-
12377 vance and rhetorical status. *Computational linguistics* 28(4), 409–445.
- 12378 Thomas, M., B. Pang, and L. Lee (2006). Get out the vote: Determining support or opposi-
12379 tion from Congressional floor-debate transcripts. In *Proceedings of Empirical Methods for
12380 Natural Language Processing (EMNLP)*, pp. 327–335.
- 12381 Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal
12382 Statistical Society. Series B (Methodological)*, 267–288.
- 12383 Titov, I. and J. Henderson (2007). Constituent parsing with incremental sigmoid belief
12384 networks. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 632–
12385 639.
- 12386 Toutanova, K., D. Klein, C. D. Manning, and Y. Singer (2003). Feature-rich part-of-speech
12387 tagging with a cyclic dependency network. In *Proceedings of the North American Chapter
12388 of the Association for Computational Linguistics (NAACL)*.
- 12389 Trivedi, R. and J. Eisenstein (2013). Discourse connectors for latent subjectivity in senti-
12390 ment analysis. In *Proceedings of the North American Chapter of the Association for Compu-
12391 tational Linguistics (NAACL)*, pp. 808–813.
- 12392 Tromble, R. W. and J. Eisner (2006). A fast finite-state relaxation method for enforcing
12393 global constraints on sequence decoding. In *Proceedings of the North American Chapter of
12394 the Association for Computational Linguistics (NAACL)*, pp. 423.

- 12395 Tsochantaridis, I., T. Hofmann, T. Joachims, and Y. Altun (2004). Support vector machine
12396 learning for interdependent and structured output spaces. In *Proceedings of the twenty-*
12397 *first international conference on Machine learning*, pp. 104. ACM.
- 12398 Tsvetkov, Y., M. Faruqui, W. Ling, G. Lample, and C. Dyer (2015). Evaluation of word
12399 vector representations by subspace alignment. In *Proceedings of Empirical Methods for*
12400 *Natural Language Processing (EMNLP)*, pp. 2049–2054.
- 12401 Tu, Z., Z. Lu, Y. Liu, X. Liu, and H. Li (2016). Modeling coverage for neural machine
12402 translation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 76–
12403 85.
- 12404 Turian, J., L. Ratinov, and Y. Bengio (2010). Word representations: a simple and general
12405 method for semi-supervised learning. In *Proceedings of the Association for Computational*
12406 *Linguistics (ACL)*, pp. 384–394.
- 12407 Turing, A. M. (2009). Computing machinery and intelligence. In *Parsing the Turing Test*,
12408 pp. 23–65. Springer.
- 12409 Turney, P. D. and P. Pantel (2010). From frequency to meaning: Vector space models of
12410 semantics. *Journal of Artificial Intelligence Research* 37, 141–188.
- 12411 Tutin, A. and R. Kittredge (1992). Lexical choice in context: generating procedural texts.
12412 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.
12413 763–769.
- 12414 Twain, M. (1997). *A Tramp Abroad*. New York: Penguin.
- 12415 Tzeng, E., J. Hoffman, T. Darrell, and K. Saenko (2015). Simultaneous deep transfer across
12416 domains and tasks. In *Proceedings of the IEEE International Conference on Computer Vision*,
12417 pp. 4068–4076.
- 12418 Usunier, N., D. Buffoni, and P. Gallinari (2009). Ranking with ordered weighted pairwise
12419 classification. In *Proceedings of the International Conference on Machine Learning (ICML)*,
12420 pp. 1057–1064.
- 12421 Uthus, D. C. and D. W. Aha (2013). The ubuntu chat corpus for multiparicipant chat
12422 analysis. In *AAAI Spring Symposium: Analyzing Microtext*, Volume 13, pp. 01.
- 12423 Utiyama, M. and H. Isahara (2001). A statistical model for domain-independent text seg-
12424 mentation. In *Proceedings of the 39th Annual Meeting on Association for Computational*
12425 *Linguistics*, pp. 499–506. Association for Computational Linguistics.

- 12426 Utiyama, M. and H. Isahara (2007). A comparison of pivot methods for phrase-based
12427 statistical machine translation. In *Human Language Technologies 2007: The Conference of
12428 the North American Chapter of the Association for Computational Linguistics; Proceedings of
12429 the Main Conference*, pp. 484–491.
- 12430 Uzuner, Ö., X. Zhang, and T. Sibanda (2009). Machine learning and rule-based approaches
12431 to assertion classification. *Journal of the American Medical Informatics Association* 16(1),
12432 109–115.
- 12433 Vadas, D. and J. R. Curran (2011). Parsing noun phrases in the penn treebank. *Computa-
12434 tional Linguistics* 37(4), 753–809.
- 12435 Van Eynde, F. (2006). NP-internal agreement and the structure of the noun phrase. *Journal
12436 of Linguistics* 42(1), 139–186.
- 12437 Van Gael, J., A. Vlachos, and Z. Ghahramani (2009). The infinite hmm for unsuper-
12438 vised pos tagging. In *Proceedings of Empirical Methods for Natural Language Processing
12439 (EMNLP)*, pp. 678–687.
- 12440 Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and
12441 I. Polosukhin (2017). Attention is all you need. In *Neural Information Processing Systems
12442 (NIPS)*, pp. 6000–6010.
- 12443 Velldal, E., L. Øvrelid, J. Read, and S. Oepen (2012). Speculation and negation: Rules,
12444 rankers, and the role of syntax. *Computational linguistics* 38(2), 369–410.
- 12445 Versley, Y. (2011). Towards finer-grained tagging of discourse connectives. In *Proceedings
12446 of the Workshop Beyond Semantics: Corpus-based Investigations of Pragmatic and Discourse
12447 Phenomena*, pp. 2–63.
- 12448 Vilain, M., J. Burger, J. Aberdeen, D. Connolly, and L. Hirschman (1995). A model-
12449 theoretic coreference scoring scheme. In *Proceedings of the 6th conference on Message
12450 understanding*, pp. 45–52. Association for Computational Linguistics.
- 12451 Vincent, P., H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol (2010). Stacked de-
12452 noising autoencoders: Learning useful representations in a deep network with a local
12453 denoising criterion. *Journal of Machine Learning Research* 11(Dec), 3371–3408.
- 12454 Vincze, V., G. Szarvas, R. Farkas, G. Móra, and J. Csirik (2008). The bioscope corpus:
12455 biomedical texts annotated for uncertainty, negation and their scopes. *BMC bioinformat-
12456 ics* 9(11), S9.
- 12457 Vinyals, O., A. Toshev, S. Bengio, and D. Erhan (2015). Show and tell: A neural image cap-
12458 tion generator. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference
12459 on*, pp. 3156–3164. IEEE.

- 12460 Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum
12461 decoding algorithm. *IEEE transactions on Information Theory* 13(2), 260–269.
- 12462 Voll, K. and M. Taboada (2007). Not all words are created equal: Extracting semantic
12463 orientation as a function of adjective relevance. In *Proceedings of Australian Conference
12464 on Artificial Intelligence*.
- 12465 Wager, S., S. Wang, and P. S. Liang (2013). Dropout training as adaptive regularization. In
12466 *Neural Information Processing Systems (NIPS)*, pp. 351–359.
- 12467 Wainwright, M. J. and M. I. Jordan (2008). Graphical models, exponential families, and
12468 variational inference. *Foundations and Trends® in Machine Learning* 1(1-2), 1–305.
- 12469 Walker, M. A. (2000). An application of reinforcement learning to dialogue strategy selec-
12470 tion in a spoken dialogue system for email. *Journal of Artificial Intelligence Research* 12,
12471 387–416.
- 12472 Walker, M. A., J. E. Cahn, and S. J. Whittaker (1997). Improvising linguistic style: Social
12473 and affective bases for agent personality. In *Proceedings of the first international conference
12474 on Autonomous agents*, pp. 96–105. ACM.
- 12475 Wang, C., N. Xue, and S. Pradhan (2015). A Transition-based Algorithm for AMR Parsing.
12476 In *Proceedings of the North American Chapter of the Association for Computational Linguistics
12477 (NAACL)*, pp. 366–375.
- 12478 Wang, H., T. Onishi, K. Gimpel, and D. McAllester (2017). Emergent predication structure
12479 in hidden state vectors of neural readers. In *Proceedings of the 2nd Workshop on Represen-
12480 tation Learning for NLP*, pp. 26–36.
- 12481 Weaver, W. (1955). Translation. *Machine translation of languages* 14, 15–23.
- 12482 Webber, B. (2004, sep). D-LTAG: extending lexicalized TAG to discourse. *Cognitive Sci-
12483 ence* 28(5), 751–779.
- 12484 Webber, B., M. Egg, and V. Kordoni (2012). Discourse structure and language technology.
12485 *Journal of Natural Language Engineering* 1.
- 12486 Webber, B. and A. Joshi (2012). Discourse structure and computation: past, present and
12487 future. In *Proceedings of the ACL-2012 Special Workshop on Rediscovering 50 Years of Dis-
12488 coveries*, pp. 42–54. Association for Computational Linguistics.
- 12489 Wei, G. C. and M. A. Tanner (1990). A monte carlo implementation of the em algorithm
12490 and the poor man’s data augmentation algorithms. *Journal of the American Statistical
12491 Association* 85(411), 699–704.

- 12492 Weinberger, K., A. Dasgupta, J. Langford, A. Smola, and J. Attenberg (2009). Feature
12493 hashing for large scale multitask learning. In *Proceedings of the International Conference*
12494 *on Machine Learning (ICML)*, pp. 1113–1120.
- 12495 Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language
12496 communication between man and machine. *Communications of the ACM* 9(1), 36–45.
- 12497 Wellner, B. and J. Pustejovsky (2007). Automatically identifying the arguments of dis-
12498 course connectives. In *Proceedings of Empirical Methods for Natural Language Processing*
12499 (*EMNLP*), pp. 92–101.
- 12500 Wen, T.-H., M. Gasic, N. Mrkšić, P.-H. Su, D. Vandyke, and S. Young (2015). Semantically
12501 conditioned lstm-based natural language generation for spoken dialogue systems. In
12502 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1711–1721.
- 12503 Weston, J., S. Bengio, and N. Usunier (2011). Wsabie: Scaling up to large vocabulary image
12504 annotation. In *IJCAI*, Volume 11, pp. 2764–2770.
- 12505 Wiebe, J., T. Wilson, and C. Cardie (2005). Annotating expressions of opinions and emo-
12506 tions in language. *Language resources and evaluation* 39(2), 165–210.
- 12507 Wieting, J., M. Bansal, K. Gimpel, and K. Livescu (2015). Towards universal paraphrastic
12508 sentence embeddings. *arXiv preprint arXiv:1511.08198*.
- 12509 Wieting, J., M. Bansal, K. Gimpel, and K. Livescu (2016). CHARAGRAM: Embedding
12510 words and sentences via character n-grams. In *Proceedings of Empirical Methods for Nat-*
12511 *ural Language Processing (EMNLP)*, pp. 1504–1515.
- 12512 Williams, J. D. and S. Young (2007). Partially observable markov decision processes for
12513 spoken dialog systems. *Computer Speech & Language* 21(2), 393–422.
- 12514 Williams, P., R. Sennrich, M. Post, and P. Koehn (2016). Syntax-based statistical machine
12515 translation. *Synthesis Lectures on Human Language Technologies* 9(4), 1–208.
- 12516 Wilson, T., J. Wiebe, and P. Hoffmann (2005). Recognizing contextual polarity in phrase-
12517 level sentiment analysis. In *Proceedings of Empirical Methods for Natural Language Pro-*
12518 *cessing (EMNLP)*, pp. 347–354.
- 12519 Winograd, T. (1972). Understanding natural language. *Cognitive psychology* 3(1), 1–191.
- 12520 Wiseman, S., A. M. Rush, and S. M. Shieber (2016). Learning global features for corefer-
12521 ence resolution. In *Proceedings of the North American Chapter of the Association for Compu-*
12522 *tational Linguistics (NAACL)*, pp. 994–1004.

- 12523 Wiseman, S., S. Shieber, and A. Rush (2017). Challenges in data-to-document generation.
 12524 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 2253–
 12525 2263.
- 12526 Wiseman, S. J., A. M. Rush, S. M. Shieber, and J. Weston (2015). Learning anaphoricity and
 12527 antecedent ranking features for coreference resolution. In *Proceedings of the Association
 12528 for Computational Linguistics (ACL)*.
- 12529 Wolf, F. and E. Gibson (2005). Representing discourse coherence: A corpus-based study.
 12530 *Computational Linguistics* 31(2), 249–287.
- 12531 Wolfe, T., M. Dredze, and B. Van Durme (2017). Pocket knowledge base population. In
 12532 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 305–310.
- 12533 Wong, Y. W. and R. Mooney (2007). Generation by inverting a semantic parser that uses
 12534 statistical machine translation. In *Proceedings of the North American Chapter of the Associa-
 12535 tion for Computational Linguistics (NAACL)*, pp. 172–179.
- 12536 Wong, Y. W. and R. J. Mooney (2006). Learning for semantic parsing with statistical ma-
 12537 chine translation. In *Proceedings of the North American Chapter of the Association for Com-
 12538 putational Linguistics (NAACL)*, pp. 439–446.
- 12539 Wu, B. Y. and K.-M. Chao (2004). *Spanning trees and optimization problems*. CRC Press.
- 12540 Wu, D. (1997). Stochastic inversion transduction grammars and bilingual parsing of par-
 12541 allel corpora. *Computational linguistics* 23(3), 377–403.
- 12542 Wu, F. and D. S. Weld (2010). Open information extraction using wikipedia. In *Proceedings
 12543 of the Association for Computational Linguistics (ACL)*, pp. 118–127.
- 12544 Wu, X., R. Ward, and L. Bottou (2018). Wngrad: Learn the learning rate in gradient de-
 12545 scent. *arXiv preprint arXiv:1803.02865*.
- 12546 Wu, Y., M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao,
 12547 Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws,
 12548 Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young,
 12549 J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean (2016).
 12550 Google’s neural machine translation system: Bridging the gap between human and ma-
 12551 chine translation. *CoRR abs/1609.08144*.
- 12552 Xia, F. (2000). The part-of-speech tagging guidelines for the penn chinese treebank (3.0).
 12553 Technical report, University of Pennsylvania Institute for Research in Cognitive Science.
- 12554 Xu, K., J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio
 12555 (2015). Show, attend and tell: Neural image caption generation with visual attention.
 12556 In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 2048–2057.

- 12557 Xu, W., X. Liu, and Y. Gong (2003). Document clustering based on non-negative matrix
12558 factorization. In *SIGIR*, pp. 267–273. ACM.
- 12559 Xu, Y., L. Mou, G. Li, Y. Chen, H. Peng, and Z. Jin (2015). Classifying relations via long
12560 short term memory networks along shortest dependency paths. In *Proceedings of Empirical
12561 Methods for Natural Language Processing (EMNLP)*, pp. 1785–1794.
- 12562 Xuan Bach, N., N. L. Minh, and A. Shimazu (2012). A reranking model for discourse seg-
12563 mentation using subtree features. In *Proceedings of the Special Interest Group on Discourse
12564 and Dialogue (SIGDIAL)*.
- 12565 Xue, N. et al. (2003). Chinese word segmentation as character tagging. *Computational
12566 Linguistics and Chinese Language Processing* 8(1), 29–48.
- 12567 Xue, N., H. T. Ng, S. Pradhan, R. Prasad, C. Bryant, and A. T. Rutherford (2015). The
12568 CoNLL-2015 shared task on shallow discourse parsing. In *Proceedings of the Conference
12569 on Natural Language Learning (CoNLL)*.
- 12570 Xue, N., H. T. Ng, S. Pradhan, A. Rutherford, B. L. Webber, C. Wang, and H. Wang (2016).
12571 Conll 2016 shared task on multilingual shallow discourse parsing. In *CoNLL Shared
12572 Task*, pp. 1–19.
- 12573 Yamada, H. and Y. Matsumoto (2003). Statistical dependency analysis with support vector
12574 machines. In *Proceedings of IWPT*, Volume 3, pp. 195–206.
- 12575 Yamada, K. and K. Knight (2001). A syntax-based statistical translation model. In *Proceed-
12576 ings of the 39th Annual Meeting on Association for Computational Linguistics*, pp. 523–530.
12577 Association for Computational Linguistics.
- 12578 Yang, B. and C. Cardie (2014). Context-aware learning for sentence-level sentiment anal-
12579 ysis with posterior regularization. In *Proceedings of the Association for Computational Lin-
12580 guistics (ACL)*.
- 12581 Yang, Y., M.-W. Chang, and J. Eisenstein (2016). Toward socially-infused information ex-
12582 traction: Embedding authors, mentions, and entities. In *Proceedings of Empirical Methods
12583 for Natural Language Processing (EMNLP)*.
- 12584 Yang, Y. and J. Eisenstein (2013). A log-linear model for unsupervised text normalization.
12585 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12586 Yang, Y. and J. Eisenstein (2015). Unsupervised multi-domain adaptation with feature em-
12587 beddings. In *Proceedings of the North American Chapter of the Association for Computational
12588 Linguistics (NAACL)*.

- 12589 Yang, Y., W.-t. Yih, and C. Meek (2015). WikiQA: A challenge dataset for open-domain
 12590 question answering. In *Proceedings of Empirical Methods for Natural Language Processing*
 12591 (*EMNLP*), pp. 2013–2018.
- 12592 Yannakoudakis, H., T. Briscoe, and B. Medlock (2011). A new dataset and method for
 12593 automatically grading esol texts. In *Proceedings of the 49th Annual Meeting of the Association*
 12594 *for Computational Linguistics: Human Language Technologies-Volume 1*, pp. 180–189.
 12595 Association for Computational Linguistics.
- 12596 Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods.
 12597 In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 189–196.
 12598 Association for Computational Linguistics.
- 12599 Yee, L. C. and T. Y. Jones (2012, March). Apple ceo in china mission to clear up problems.
 12600 *Reuters*. retrieved on March 26, 2017.
- 12601 Yi, Y., C.-Y. Lai, S. Petrov, and K. Keutzer (2011, October). Efficient parallel cky parsing on
 12602 gpus. In *Proceedings of the 12th International Conference on Parsing Technologies*, Dublin,
 12603 Ireland, pp. 175–185. Association for Computational Linguistics.
- 12604 Yu, C.-N. J. and T. Joachims (2009). Learning structural svms with latent variables. In
 12605 *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1169–1176.
- 12606 Yu, F. and V. Koltun (2016). Multi-scale context aggregation by dilated convolutions. In
 12607 *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 12608 Zaidan, O. F. and C. Callison-Burch (2011). Crowdsourcing translation: Professional quality
 12609 from non-professionals. In *Proceedings of the Association for Computational Linguistics*
 12610 (*ACL*), pp. 1220–1229.
- 12611 Zaremba, W., I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.
- 12612 Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.
- 12613 Zelenko, D., C. Aone, and A. Richardella (2003). Kernel methods for relation extraction.
 12614 *The Journal of Machine Learning Research* 3, 1083–1106.
- 12615 Zelle, J. M. and R. J. Mooney (1996). Learning to parse database queries using inductive
 12616 logic programming. In *Proceedings of the National Conference on Artificial Intelligence*
 12617 (*AAAI*), pp. 1050–1055.
- 12618 Zeng, D., K. Liu, S. Lai, G. Zhou, and J. Zhao (2014). Relation classification via convolutional
 12619 deep neural network. In *Proceedings of the International Conference on Computational*
 12620 *Linguistics (COLING)*, pp. 2335–2344.

- 12623 Zettlemoyer, L. S. and M. Collins (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of UAI*.
- 12624
- 12625 Zhang, X., J. Zhao, and Y. LeCun (2015). Character-level convolutional networks for text classification. In *Neural Information Processing Systems (NIPS)*, pp. 649–657.
- 12626
- 12627 Zhang, Y. and S. Clark (2008). A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 562–571.
- 12628
- 12629
- 12630 Zhang, Y., T. Lei, R. Barzilay, T. Jaakkola, and A. Globerson (2014). Steps to excellence: Simple inference with refined scoring of dependency trees. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 197–207.
- 12631
- 12632
- 12633 Zhang, Y. and J. Nivre (2011). Transition-based dependency parsing with rich non-local features. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 188–193.
- 12634
- 12635 Zhou, J. and W. Xu (2015). End-to-end learning of semantic role labeling using recurrent neural networks. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1127–1137.
- 12636
- 12637
- 12638 Zhu, J., Z. Nie, X. Liu, B. Zhang, and J.-R. Wen (2009). Statsnowball: a statistical approach to extracting entity relationships. In *Proceedings of the Conference on World-Wide Web (WWW)*, pp. 101–110.
- 12639
- 12640
- 12641 Zhu, X., Z. Ghahramani, and J. D. Lafferty (2003). Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 912–919.
- 12642
- 12643
- 12644 Zhu, X. and A. B. Goldberg (2009). Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning* 3(1), 1–130.
- 12645
- 12646 Zipf, G. K. (1949). Human behavior and the principle of least effort.
- 12647 Zirn, C., M. Niepert, H. Stuckenschmidt, and M. Strube (2011). Fine-grained sentiment analysis with structural features. In *IJCNLP*, Chiang Mai, Thailand, pp. 336–344.
- 12648
- 12649 Zou, W. Y., R. Socher, D. Cer, and C. D. Manning (2013). Bilingual word embeddings for phrase-based machine translation. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1393–1398.
- 12650
- 12651

12652 **Index**

- 12653 *K*-means, 108
12654 α -conversion, 298
12655 β -conversion, 295
12656 β -reduction, 295
12657 *n*-gram language models, 206
12658 *n*-gram, 40
12659 *F*-MEASURE, 94
12660 BLEU, 437
12661 WordNet, 86
12662 ablation test, 96
12663 absolute discounting, 142
12664 Abstract Meaning Representation
 (AMR), 311, 323
12666 abstractive summarization, 399, 466
12667 accepting path, 201
12668 accuracy, 38, 93
12669 action (reinforcement learning), 373
12670 active learning, 130
12671 AdaDelta (online optimization), 75
12672 AdaGrad (online optimization), 55, 75
12673 Adam (online optimization), 75
12674 adequacy (translation), 437
12675 adjectives, 187
12676 adjuncts (semantics), 310, 314
12677 adpositions, 188
12678 adverbs, 187
12679 adversarial networks, 125
12680 affix (morphology), 203
12681 agenda-based dialogue systems, 470
12682 agenda-based parsing, 282
12683 agent (thematic role), 312
12684 alignment, 325
12685 alignment (machine translation), 436,
 441
12687 alignment (text generation), 461
12688 Amazon Mechanical Turk, 102
12689 ambiguity, 218, 226
12690 ambiguity, attachment, 237
12691 ambiguity, complement structure, 237
12692 ambiguity, coordination scope, 237
12693 ambiguity, modifier scope, 237
12694 ambiguity, particle versus preposition,
 237
12696 anaphoric, 368
12697 anchored productions, 240
12698 animacy (semantics), 311
12699 annealing, 456
12700 antecedent (coreference), 359, 367
12701 antonymy, 86
12702 apophony, 202
12703 arc-eager dependency parsing, 273, 275
12704 arc-factored dependency parsing, 267
12705 arc-standard dependency parsing, 273
12706 area under the curve (AUC), 95
12707 argumentation, 398
12708 argumentation mining, 399
12709 arguments, 407
12710 article (syntax), 192
12711 aspect, 187
12712 aspect-based opinion mining, 83
12713 attachment ambiguity, 261

- 12714 attention mechanism, 379, 430, 449, 462
 12715 autoencoder, 353
 12716 autoencoder, denoising, 353, 412
 12717 automated theorem provers, 290
 12718 automatic differentiation, 70
 12719 auxiliary verbs, 188
 12720 average mutual information, 341
 12721 averaged perceptron, 42
 12722 backchannel, 197
 12723 backoff, 142
 12724 backpropagation, 69, 148
 12725 backpropagation through time, 148
 12726 backward recurrence, 175, 176
 12727 backward-looking center, 388
 12728 bag of words, 29
 12729 balanced *F*-MEASURE, 95
 12730 balanced test set, 93
 12731 batch learning, 41
 12732 batch normalization, 74
 12733 batch optimization, 53
 12734 Baum-Welch algorithm, 180
 12735 Bayes' rule, 478
 12736 Bayesian nonparametrics, 115, 256
 12737 beam sampling, 182
 12738 beam search, 275, 372, 453, 454
 12739 Bell number, 370
 12740 best-path algorithm, 206
 12741 bias (learning theory), 38, 139
 12742 bias-variance tradeoff, 38, 141
 12743 biconvexity, 114
 12744 bidirectional LSTM, 449
 12745 bidirectional recurrent neural network,
 178
 12746 bigrams, 40, 82
 12747 bilexical, 255
 12748 bilexical features, 270
 12750 bilinear product, 338
 12751 binarization (context-free grammar),
 219, 236
 12753 binomial distribution, 96
 12754 binomial random variable, 482
 12755 binomial test, 96
 12756 BIO notation, 194, 321
 12757 biomedical natural language processing,
 193
 12759 bipartite graph, 327
 12760 Bonferroni correction, 99
 12761 boolean semiring, 207
 12762 boosting, 62
 12763 bootstrap samples, 98
 12764 brevity penalty (machine translation),
 438
 12766 Brown clusters, 335, 340
 12767 byte-pair encoding, 351, 452
 12768 c-command, 361
 12769 case marking, 192, 227
 12770 Catalan number, 233
 12771 cataphora, 360
 12772 center embedding, 215
 12773 centering theory, 363, 388
 12774 chain FSA, 213
 12775 chain rule of probability, 477
 12776 chance agreement, 102
 12777 character-level language models, 153
 12778 chart parsing, 234
 12779 chatbots, 472
 12780 Chomsky Normal Form (CNF), 219
 12781 Chu-Liu-Edmonds algorithm, 268
 12782 CKY algorithm, 234
 12783 class imbalance, 93
 12784 classification weights, 29
 12785 closed-vocabulary, 153
 12786 closure (regular languages), 200
 12787 cloze question answering, 429
 12788 cluster ranking, 371
 12789 clustering, 108
 12790 co-training, 120
 12791 coarse-to-fine attention, 465
 12792 code switching, 190, 196
 12793 Cohen's Kappa, 102

- 12794 coherence, 402
 12795 cohesion, 385
 12796 collapsed Gibbs sampling, 128
 12797 collective entity linking, 412, 413
 12798 collocation extraction, 352
 12799 collocation features, 87
 12800 combinatorial optimization, 19
 12801 combinatory categorial grammar, 228
 12802 complement clause, 221
 12803 complement event (probability), 476
 12804 composition (CCG), 229
 12805 compositional vector grammars, 397
 12806 compositionality, 18, 21, 350
 12807 computation graph, 62, 69
 12808 computational linguistics (versus
 natural language processing), 13
 12809 computational social science, 17
 12811 concept (AMR), 323
 12812 conditional independence, 164, 480
 12813 conditional log-likelihood, 66
 12814 conditional probability, 50, 477
 12815 conditional probability distribution, 481
 12816 conditional random field, 172
 12817 confidence interval, 98
 12818 configuration (transition-based parsing),
 273
 12819 connected (graph theory), 325
 12821 consistency (logic), 293
 12822 constants (logic), 288
 12823 constituents, 220
 12824 constrained optimization, 47, 319
 12825 constraint-driven learning, 130
 12826 constraints, 319
 12827 content selection (text generation), 459
 12828 content words, 188
 12829 context vector (attentional neural
 translation), 450
 12831 context-free grammars (CFGs), 216
 12832 context-free languages, 215, 216
 12833 context-sensitive languages, 226
 12834 continuous bag-of-words (CBOW), 343
 12835 contradiction, 354
 12836 conversational turns, 197
 12837 convex optimization, 53
 12838 convexity, 44, 72, 303, 482, 485
 12839 convolutional neural networks, 67, 76,
 82, 153, 180, 195, 419
 12841 cooperative principle, 359
 12842 coordinate ascent, 114
 12843 coordinating conjunctions, 188
 12844 coordinating discourse relations, 395
 12845 copula, 187, 225, 264
 12846 coreference resolution, 355, 359
 12847 coreferent, 359
 12848 cosine similarity, 347, 386
 12849 cost-augmented decoding, 49, 171
 12850 coverage (summarization), 399
 12851 coverage loss, 467
 12852 critical point, 72, 485
 12853 cross-document coreference resolution,
 410
 12854 cross-entropy, 66, 420
 12855 cross-serial dependencies, 227
 12856 cross-validation, 39
 12857 crowdsourcing, 102
 12859 cumulative probability distribution, 97
 12860 dead neurons, 65
 12861 decidability (logic), 293
 12862 decision trees, 62
 12863 deep learning, 61
 12864 deep LSTM, 448
 12865 definiteness, 193
 12866 delta function, 37
 12867 denotation (semantics), 288
 12868 dependency grammar, 261
 12869 dependency graph, 262
 12870 dependency parse, 261
 12871 dependency path, 87, 318, 417
 12872 dependent, 262
 12873 derivation (context-free languages), 217
 12874 derivation (parsing), 272, 278

- 12875 derivation (semantic parsing), 296, 300
 12876 derivational ambiguity, 230
 12877 derivational morphology, 202
 12878 determiner, 189
 12879 determiner phrase, 223
 12880 deterministic FSA, 202
 12881 development set, 39, 93
 12882 dialogue acts, 102, 197, 474
 12883 dialogue management, 470
 12884 dialogue systems, 137, 469
 12885 digital humanities, 17, 81
 12886 dilated convolution, 77, 195
 12887 Dirichlet distribution, 127
 12888 discounting (language models), 142
 12889 discourse, 385
 12890 discourse connectives, 391
 12891 discourse depth, 400
 12892 discourse depth tree, 401
 12893 discourse parsing, 390
 12894 discourse relations, 355, 390
 12895 discourse segment, 385
 12896 discourse sense classification, 392
 12897 discourse unit, 395
 12898 discrete random variable, 480
 12899 discriminative learning, 40
 12900 disjoint events (probability), 476
 12901 distant supervision, 130, 422, 423
 12902 distributed semantics, 334
 12903 distributional hypothesis, 333, 334
 12904 distributional semantics, 22, 334
 12905 distributional statistics, 87, 255, 334
 12906 document frequency, 411
 12907 domain adaptation, 107, 123
 12908 dropout, 71, 149
 12909 dual decomposition, 321
 12910 dynamic computation graphs, 70
 12911 dynamic oracle, 280
 12912 dynamic programming, 159
 12913 dynamic semantics, 305, 390
 12914 E-step (expectation-maximization), 111
 12915 early stopping, 43, 75
 12916 early update, 280
 12917 easy-first parsing, 283
 12918 edit distance, 209, 439
 12919 effective count (language models), 141
 12920 elementary discourse units, 395
 12921 elementwise nonlinearity, 63
 12922 Elman unit, 147
 12923 ELMo (embeddings from language models), 349
 12924 embedding, 177, 343
 12926 emission features, 158
 12927 emotion, 84
 12928 empirical Bayes, 128
 12929 empty string, 200
 12930 encoder-decoder, 446
 12931 encoder-decoder model, 353, 462
 12932 ensemble, 322
 12933 ensemble learning, 448
 12934 ensemble methods, 62
 12935 entailment, 293, 354
 12936 entities, 407
 12937 entity embeddings, 411
 12938 entity grid, 389
 12939 entity linking, 359, 407, 409, 420
 12940 entropy, 57, 111
 12941 estimation, 482
 12942 EuroParl corpus, 439
 12943 event, 425
 12944 event (probability), 475
 12945 event coreference, 425
 12946 event detection, 425
 12947 event semantics, 309
 12948 events, 407
 12949 evidentiality, 192, 427
 12950 exchange clustering, 342
 12951 expectation, 481
 12952 expectation maximization, 109, 143
 12953 expectation semiring, 215
 12954 expectation-maximization, in machine translation, 443
 12955

- | | | | |
|-------|---|-------|--|
| 12956 | explicit semantic analysis, 336 | 12996 | FrameNet, 314 |
| 12957 | exploding gradients, 149 | 12997 | frames, 314 |
| 12958 | extra-propositional semantics, 427 | 12998 | Frobenius norm, 71 |
| 12959 | extractive question-answering, 430 | 12999 | function (first-order logic), 292 |
| 12960 | extractive summarization, 399 | 13000 | function words, 188 |
| 12961 | extrinsic evaluation, 151, 347 | 13001 | functional margin, 47 |
| 12962 | factoid questions, 326 | 13002 | functional segmentation, 385, 387 |
| 12963 | factoids, 428 | 13003 | garden path sentence, 156 |
| 12964 | factor graph, 173 | 13004 | gate (neural networks), 66, 449 |
| 12965 | factuality, 427 | 13005 | gazetteer, 417 |
| 12966 | false discovery rate, 99 | 13006 | gazetteers, 365 |
| 12967 | False negative, 93 | 13007 | gazzeteers, 194 |
| 12968 | False positive, 93 | 13008 | generalization, 43 |
| 12969 | false positive, 479 | 13009 | generalized linear models, 57 |
| 12970 | false positive rate, 95, 478 | 13010 | generative model, 33, 243 |
| 12971 | feature co-adaptation, 71 | 13011 | generative models, 371 |
| 12972 | feature function, 30, 39 | 13012 | generative process, 143 |
| 12973 | feature hashing, 92 | 13013 | generic referents, 364 |
| 12974 | feature noising, 72 | 13014 | geometric margin, 47 |
| 12975 | feature selection, 56 | 13015 | Gibbs sampling, 127, 413 |
| 12976 | features, 18 | 13016 | gloss, 137, 189, 437 |
| 12977 | feedforward neural network, 64 | 13017 | government and binding theory, 361 |
| 12978 | fine-tuned word embeddings, 349 | 13018 | gradient, 45 |
| 12979 | finite state acceptor (FSA), 201 | 13019 | gradient clipping, 74 |
| 12980 | finite state automata, 201 | 13020 | gradient descent, 53 |
| 12981 | finite state composition, 212 | 13021 | Gram matrix, 418 |
| 12982 | finite state transducers, 204, 209 | 13022 | grammar induction, 249 |
| 12983 | first-order logic, 291 | 13023 | grammaticality, 402 |
| 12984 | fluency (translation), 437 | 13024 | graph-based dependency parsing, 267 |
| 12985 | fluent, 137 | 13025 | graphical model, 164 |
| 12986 | focus, 324 | 13026 | graphics processing units (GPUs), 180, 195 |
| 12987 | formal language theory, 199 | 13027 | grid search, 38 |
| 12988 | forward recurrence, 174 | 13028 | |
| 12989 | forward variable, 176 | 13029 | Hamming cost, 171 |
| 12990 | forward variables, 174 | 13030 | Hansards corpus, 439 |
| 12991 | forward-backward algorithm, 175, 214, 247 | 13031 | hanzi, 89 |
| 12993 | forward-looking centers, 388 | 13032 | hard expectation-maximization, 114 |
| 12994 | frame, 469 | 13033 | head, 262, 417 |
| 12995 | frame elements, 314 | 13034 | head percolation rules, 252 |

- 13035 head rules, 261
 13036 head word, 220, 261, 365
 13037 head words, 252
 13038 hedging, 427
 13039 held-out data, 151
 13040 Hessian matrix, 54
 13041 hidden Markov models, 164
 13042 hidden variable perceptron, 215
 13043 hierarchical clustering, 340
 13044 hierarchical recurrent network, 473
 13045 hierarchical softmax, 147, 345
 13046 hierarchical topic segmentation, 387
 13047 highway network, 66
 13048 hinge loss, 44
 13049 holonymy, 87
 13050 homonym, 85
 13051 human computation, 103
 13052 hypergraph, 398
 13053 hypernymy, 87
 13054 hyperparameter, 38
 13055 hyponymy, 87
- 13056 illocutionary force, 197
 13057 implicit discourse relations, 392
 13058 importance sampling, 456
 13059 importance score, 456
 13060 incremental expectation maximization, 114
 13061 incremental perceptron, 280, 372
 13063 independent and identically distributed (IID), 32
 13064 indicator function, 37
 13066 indicator random variable, 480
 13067 inference, 157
 13068 inference (logic), 287
 13069 inference rules, 290
 13070 inflection point, 486
 13071 inflectional affixes, 90
 13072 inflectional morphology, 187, 202, 210
 13073 information extraction, 407
 13074 information retrieval, 17, 421
- 13075 initiative (dialogue systems), 470
 13076 input word embeddings, 147
 13077 inside recurrence, 243, 244, 248
 13078 inside-outside algorithm, 247, 256
 13079 instance (AMR), 323
 13080 instance labels, 32
 13081 integer linear program, 432, 468
 13082 integer linear programming, 319, 370, 401, 413
 13084 inter-annotator agreement, 102
 13085 interjections, 187
 13086 interlingua, 436
 13087 interpolated n -gram language model, 207
 13089 interpolation, 143
 13090 interval algebra, 426
 13091 intrinsic evaluation, 151, 347
 13092 inverse document frequency, 411
 13093 inverse relation (AMR), 324
 13094 inversion (finite state), 211
 13095 irrealis, 82
- 13096 Jeffreys-Perks law, 141
 13097 Jensen's inequality, 111
 13098 joint probabilities, 481
 13099 joint probability, 32, 50
- 13100 Kalman smoother, 182
 13101 Katz backoff, 142
 13102 kernel function, 418
 13103 kernel methods, 62
 13104 kernel support vector machine, 62, 418
 13105 Kleene star, 200
 13106 knapsack problem, 401
 13107 knowledge base, 407
 13108 knowledge base population, 420
- 13109 L-BFGS, 54
 13110 label bias problem, 279
 13111 label propagation, 122, 132
 13112 labeled dependencies, 263
 13113 labeled precision, 238

- 13114 labeled recall, 238
 13115 Lagrange multiplier, 487
 13116 Lagrangian, 487
 13117 Lagrangian dual, 487
 13118 lambda calculus, 295
 13119 lambda expressions, 295
 13120 language model, 138
 13121 language models, 16
 13122 Laplace smoothing, 38, 141
 13123 large margin classification, 46
 13124 latent conditional random fields, 303
 13125 latent Dirichlet allocation, 386
 13126 latent semantic analysis, 336, 338
 13127 latent variable, 110, 214, 302, 423, 436
 13128 latent variable perceptron, 303, 368
 13129 layer normalization, 75, 451
 13130 leaky ReLU, 65
 13131 learning to search, 261, 281, 374
 13132 least squares, 84
 13133 leave-one-out, 39
 13134 lemma, 85
 13135 lemma (lexical semantics), 210
 13136 lemmatization, 90
 13137 Levenshtein edit distance, 209
 13138 lexical entry, 296
 13139 lexical features, 61
 13140 lexical semantics, 85
 13141 lexical unit (frame semantics), 314
 13142 lexicalization, 252
 13143 lexicalization (text generation), 459
 13144 lexicalized tree-adjoining grammar for discourse (D-LTAG), 391
 13145 lexicon, 296
 13147 lexicon (CCG), 229
 13148 lexicon-based classification, 84
 13149 lexicon-based sentiment analysis, 82
 13150 Lidstone smoothing, 141
 13151 light verb, 325
 13152 likelihood, 478
 13153 linear regression, 84
 13154 linear separability, 41
 13155 linearization, 469
 13156 literal character, 200
 13157 local minimum, 486
 13158 local optimum, 114
 13159 locally-normalized objective, 279
 13160 log-bilinear language model, 351
 13161 logistic function, 57
 13162 logistic loss, 51
 13163 logistic regression, 50, 57
 13164 Long short-term memories, 147
 13165 long short-term memory, 149
 13166 long short-term memory (LSTM), 66, 191, 447
 13168 lookup layer, 67, 147
 13169 loss function, 43
 13170 LSTM, 149
 13171 LSTM-CRF, 179, 322
 13172 machine learning, 14
 13173 machine reading, 429
 13174 machine translation, 137
 13175 Macro F-MEASURE, 94
 13176 macro-reading, 408
 13177 margin, 41, 46
 13178 marginal probability distribution, 481
 13179 marginal relevance, 467
 13180 marginalize, 477
 13181 markable, 366
 13182 Markov assumption, 164
 13183 Markov blanket, 164
 13184 Markov Chain Monte Carlo (MCMC), 115, 127, 182
 13186 Markov decision process, 470
 13187 Markov random fields, 172
 13188 matrix-tree theorem, 272
 13189 max-margin Markov network, 171
 13190 max-product algorithm, 167
 13191 maximum a posteriori, 38, 483
 13192 maximum conditional likelihood, 50
 13193 maximum entropy, 57
 13194 maximum likelihood, 482

- 13195 maximum likelihood estimate, 36
 13196 maximum likelihood estimation, 32
 13197 maximum spanning tree, 268
 13198 McNemar's test, 96
 13199 meaning representation, 287
 13200 membership problem, 199
 13201 memory cell (LSTM), 149
 13202 mention (coreference resolution), 359
 13203 mention (entity), 407
 13204 mention (information extraction), 409
 13205 mention ranking, 368
 13206 mention-pair model, 367
 13207 meronymy, 87
 13208 meteor, 439
 13209 method of moments, 128
 13210 micro *F*-MEASURE, 94
 13211 micro-reading, 408
 13212 mildly context-sensitive languages, 227
 13213 minibatch, 54
 13214 minimization (FSA), 204
 13215 minimum error-rate training (MERT),
 455
 13217 minimum risk training, 455
 13218 mixed-initiative, 470
 13219 modality, 427
 13220 model, 19
 13221 model builder, 293
 13222 model checker, 293
 13223 model-theoretic semantics, 288
 13224 modeling (machine learning), 53
 13225 modifier (dependency grammar), 262
 13226 modus ponens, 290
 13227 moment-matching, 57
 13228 monomorphemic, 204
 13229 morphemes, 17, 153, 203
 13230 morphological analysis, 210
 13231 morphological generation, 210
 13232 morphological segmentation, 169
 13233 morphology, 91, 168, 202, 350, 452
 13234 morphosyntactic, 186
 13235 morphosyntactic attributes, 190
- 13236 morphotactic, 203
 13237 multi-document summarization, 468
 13238 multi-view learning, 120
 13239 multilayer perceptron, 64
 13240 multinomial distribution, 34
 13241 multinomial naïve Bayes, 34
 13242 multiple instance learning, 130, 423
 13243 multiple-choice question answering, 429
 13244 multitask learning, 130
- 13245 Naïve Bayes, 33
 13246 name dictionary, 410
 13247 named entities, 193
 13248 named entity linking, 409
 13249 named entity recognition, 179, 407, 409
 13250 named entity types, 409
 13251 narrow convolution, 77
 13252 nearest-neighbor, 62, 418
 13253 negation, 82, 427
 13254 negative sampling, 345, 346, 415
 13255 Neo-Davidsonian event semantics, 310
 13256 neural attention, 448
 13257 neural machine translation, 436
 13258 neural networks, 61, 146
 13259 NIL entity, 409
 13260 noise-contrastive estimation, 147
 13261 noisy channel model, 138, 440
 13262 nominal modifier, 223
 13263 nominals, 359
 13264 nominals (coreference), 366
 13265 non-core roles (AMR), 324
 13266 non-terminals (context-free grammars),
 217
 13267 normalization, 90
 13268 noun phrase, 14, 220
 13270 nouns, 186
 13271 NP-hard, 56, 413
 13272 nuclearity (RST), 395
 13273 nucleus (RST), 395
 13274 null hypothesis, 96
 13275 numeral (part of speech), 189

- 13276 numerical optimization, 20
 13277 offset feature, 31
 13278 one-dimensional convolution, 77
 13279 one-hot, 380
 13280 one-hot vector, 66
 13281 one-tailed p-value, 97
 13282 one-versus-all multiclass classification, 418
 13283 one-versus-one multiclass classification, 418
 13284 online expectation maximization, 114
 13285 online learning, 41, 54
 13288 ontology, 20
 13289 open information extraction, 424
 13290 open word classes, 186
 13291 opinion polarity, 81
 13292 oracle, 278, 326
 13293 oracle (learning to search), 374
 13294 orthography, 204, 212
 13295 orthonormal matrix, 73
 13296 out-of-vocabulary words, 191
 13297 outside recurrence, 244, 248
 13298 overfit, 38
 13299 overfitting, 42
 13300 overgeneration, 211, 221
 13301 parallel corpora, 439
 13302 parameters, 482
 13303 paraphrase, 354
 13304 parent annotation, 251
 13305 parsing, 217
 13306 part-of-speech, 185
 13307 part-of-speech tagging, 155
 13308 partially observable Markov decision process (POMDP), 472
 13309 partially supervised learning, 256
 13311 particle (part-of-speech), 189, 225
 13312 partition, 477
 13313 partition function, 174
 13314 parts-of-speech, 17
 13315 passive-aggressive, 487
 13316 path (finite state automata), 201
 13317 Penn Discourse Treebank (PDTB), 391
 13318 Penn Treebank, 152, 169, 190, 220, 246
 13319 perceptron, 41
 13320 perplexity, 152
 13321 phonology, 204
 13322 phrase (syntax), 220
 13323 phrase-structure grammar, 220
 13324 pivot features, 124
 13325 pointwise mutual information, 338
 13326 policy, 373, 471
 13327 policy (search), 280
 13328 policy gradient, 374
 13329 polysemous, 86
 13330 pooling, 381
 13331 pooling (convolution), 77, 381, 462
 13332 positional encodings, 451
 13333 positive pointwise mutual information, 339
 13334 posterior, 478
 13336 power law, 14
 13337 pragmatics, 359
 13338 pre-trained word representations, 349
 13339 precision, 94, 478
 13340 precision-at- k , 95, 404
 13341 precision-recall curve, 422
 13342 precision-recall curves, 95
 13343 predicate, 407
 13344 predicative adjectives, 225
 13345 predictive likelihood, 115
 13346 prepositional phrase, 14, 225
 13347 primal form, 487
 13348 principle of compositionality, 294
 13349 prior, 478
 13350 prior expectation, 483
 13351 probabilistic context-free grammars (PCFGs), 243
 13352 probabilistic models, 482
 13354 probabilistic topic model, 413
 13355 probability density function, 481

- 13356 probability distribution, 480
 13357 probability mass function, 97, 481
 13358 probability simplex, 33
 13359 processes, 426
 13360 production rules, 217
 13361 productivity, 203
 13362 projection function, 124
 13363 projectivity, 265
 13364 pronominal anaphora resolution, 359
 13365 pronoun, 188
 13366 PropBank, 314
 13367 proper nouns, 187
 13368 property (logic), 291
 13369 proposal distribution, 456
 13370 proposition, 426
 13371 propositions, 288, 289
 13372 prosody, 197
 13373 proto-roles, 313
 13374 pseudo-projective dependency parsing, 275
 13375 pumping lemma, 215
 13377 pushdown automata, 217
 13378 pushdown automaton, 258
 13379 quadratic program, 47
 13380 quantifier, 291
 13381 quantifier, existential, 292
 13382 quantifier, universal, 292
 13383 quasi-Newton optimization, 54
 13384 question answering, 353, 409
 13385 random outcomes, 475
 13386 ranking, 410
 13387 ranking loss, 410
 13388 recall, 94, 478
 13389 recall-at- k , 404
 13390 receiver operating characteristic (ROC), 95
 13392 rectified linear unit (ReLU), 65
 13393 recurrent neural network, 147
 13394 recurrent neural networks, 419
 13395 recursion, 14
 13396 recursive neural network, 401
 13397 recursive neural networks, 258, 351, 354
 13398 recursive production, 217
 13399 reference arguments, 329
 13400 reference resolution, 359
 13401 reference translations, 437
 13402 referent, 359
 13403 referring expression, 387
 13404 referring expressions, 359, 460
 13405 reflexive pronoun, 361
 13406 regression, 84
 13407 regular expression, 200
 13408 regular language, 200
 13409 regularization, 49
 13410 reification (events), 309
 13411 reinforcement learning, 373, 454
 13412 relation (logic), 297
 13413 relation extraction, 281, 327, 415
 13414 relations, 407
 13415 relations (information extraction), 407
 13416 relations (logic), 288
 13417 relative frequency estimate, 36, 138, 483
 13418 reranking, 257
 13419 residual networks, 66
 13420 retrofitting (word embeddings), 352
 13421 Rhetorical Structure Theory (RST), 394
 13422 rhetorical zones, 387
 13423 RIBES (translation metric), 439
 13424 ridge regression, 84
 13425 risk, 455
 13426 roll-in (reinforcement learning), 374
 13427 roll-out (reinforcement learning), 375
 13428 root (morpheme), 351
 13429 saddle point, 486
 13430 saddle points, 72
 13431 sample space, 475
 13432 satellite (RST), 395
 13433 satisfaction (logic), 293
 13434 scheduled sampling, 454

- 13435 schema, 407, 408, 424
 13436 search error, 259, 372
 13437 second-order dependency parsing, 267
 13438 second-order logic, 292
 13439 seed lexicon, 85
 13440 segmented discourse representation theory (SDRT), 390
 13441 self-attention, 450
 13443 self-training, 120
 13444 semantic, 186
 13445 semantic concordance, 88
 13446 semantic parsing, 294
 13447 semantic role, 310
 13448 Semantic role labeling, 310
 13449 semantic role labeling, 416, 424
 13450 semantic underspecification, 305
 13451 semantics, 250, 287
 13452 semi-supervised learning, 107, 117, 348
 13453 semiring algebra, 182
 13454 semiring notation, 207
 13455 semisupervised, 88
 13456 senses, 313
 13457 sentence (logic), 292
 13458 sentence compression, 467
 13459 sentence fusion, 468
 13460 sentence summarization, 466
 13461 sentiment, 81
 13462 sentiment lexicon, 32
 13463 sequence-to-sequence, 447
 13464 shift-reduce parsing, 258
 13465 shifted positive pointwise mutual information, 346
 13466 shortest-path algorithm, 205
 13468 sigmoid, 63
 13469 simplex, 127
 13470 singular value decomposition, 73
 13471 singular value decomposition (SVD), 116
 13472 singular vectors, 74
 13473 skipgram word embeddings, 344
 13474 slack variables, 48
 13475 slot filling, 420
 13476 slots (dialogue systems), 469
 13477 smooth functions, 45
 13478 smoothing, 38, 141
 13479 soft K -means, 109
 13480 softmax, 63, 146, 420
 13481 source domain, 122
 13482 source language, 435
 13483 spanning tree, 262
 13484 sparse matrix, 338
 13485 sparsity, 56
 13486 spectral learning, 129
 13487 speech acts, 197
 13488 speech recognition, 137
 13489 split constituents, 316
 13490 spurious ambiguity, 230, 272, 278, 300
 13491 squashing function, 147
 13492 squashing functions, 65
 13493 stand-off annotations, 101
 13494 Stanford Natural Language Inference corpus, 354
 13495 statistical learning theory, 42
 13496 statistical machine translation, 436
 13498 statistical significance, 96
 13499 stem, 18
 13500 stem (morphology), 203
 13501 stemmer, 90
 13502 step size, 53, 486
 13503 stochastic gradient descent, 45, 54
 13504 stoplist, 92
 13505 stopwords, 92
 13506 string (formal language theory), 199
 13507 string-to-tree translation, 446
 13508 strong compositionality criterion (RST), 397
 13509
 13510 strongly equivalent grammars, 218
 13511 structure induction, 180
 13512 structured attention, 465
 13513 structured perceptron, 170
 13514 structured prediction, 30
 13515 structured support vector machine, 171
 13516 subgradient, 45, 56

- 13517 subjectivity detection, 83
 13518 subordinating conjunctions, 188
 13519 subordinating discourse relations, 395
 13520 sum-product algorithm, 174
 13521 summarization, 137, 399
 13522 supersenses, 348
 13523 supervised machine learning, 32
 13524 support vector machine, 47
 13525 support vectors, 47
 13526 surface form, 211
 13527 surface realization, 459
 13528 synchronous context-free grammar, 445
 13529 synonymy, 86, 333
 13530 synset (synonym set), 86
 13531 synsets, 378
 13532 syntactic dependencies, 262
 13533 syntactic path, 317
 13534 syntactic-semantic grammar, 295
 13535 syntax, 185, 219, 287
- 13536 tagset, 186
 13537 tanh activation function, 65
 13538 target domain, 122
 13539 target language, 435
 13540 Targeted sentiment analysis, 83
 13541 tense, 187
 13542 terminal symbols (context-free grammars), 217
 13543 test set, 39, 107
 13545 test statistic, 96
 13546 text classification, 29
 13547 text mining, 17
 13548 text planning, 459
 13549 thematic roles, 311
 13550 third axiom of probability, 476
 13551 third-order dependency parsing, 268
 13552 TimeML, 425
 13553 tokenization, 88, 195
 13554 tokens, 34
 13555 topic models, 17
 13556 topic segmentation, 385
- 13557 trace (syntax), 230
 13558 training set, 32, 107
 13559 transduction, 200
 13560 transfer learning, 130
 13561 transformer architecture, 450
 13562 transition features, 158
 13563 transition system, 273
 13564 transition-based parsing, 233, 258
 13565 transitive closure, 369
 13566 translation error rate (TER), 439
 13567 translation model, 138
 13568 transliteration, 453
 13569 tree-adjoining grammar, 228
 13570 tree-to-string translation, 446
 13571 tree-to-tree translation, 446
 13572 treebank, 246
 13573 trellis, 160, 213
 13574 trigrams, 40
 13575 trilexical dependencies, 255
 13576 tropical semiring, 183, 207
 13577 True negative, 93
 13578 True positive, 93
 13579 true positive, 479
 13580 true positive rate, 95
 13581 truncated singular value decomposition, 338
 13582 truth conditions, 293
 13584 tuning set, 39, 93
 13585 Turing test, 15
 13586 two-tailed test, 97
 13587 type systems, 298
 13588 type-raising, 229, 298
 13589 types, 34
- 13590 unary closure, 237
 13591 unary productions, 218
 13592 underfit, 38
 13593 underflow, 33
 13594 undergeneration, 211, 221
 13595 Universal Dependencies, 186, 261
 13596 unlabeled precision, 238

- 13597 unlabeled recall, 238
13598 unseen word, 179
13599 unsupervised, 88
13600 unsupervised learning, 83, 107
13601 utterances, 197
- 13602 validation function (semantic parsing),
13603 304
13604 validity (logic), 293
13605 value function, 471
13606 value iteration, 471
13607 vanishing gradient, 65
13608 vanishing gradients, 149
13609 variable (AMR), 323
13610 variable (logic), 291
13611 variable, bound (logic), 292
13612 variable, free (logic), 291
13613 variance, 99
13614 variance (learning theory), 38
13615 variational autoencoder, 467
13616 Vauquois Pyramid, 436
13617 verb phrase, 221
13618 VerbNet, 312
13619 verbs, 187
13620 vertical Markovization, 251
- 13621 Viterbi algorithm, 158
13622 Viterbi variable, 160
13623 volition (semantics), 311
- 13624 WARP loss, 415
13625 weakly equivalent grammars, 218
13626 weight decay, 71
13627 weighted context-free grammar, 240
13628 weighted context-free grammars, 226,
13629 236
13630 weighted finite state acceptors, 205
13631 wide convolution, 77
13632 Wikification, 409
13633 Winograd schemas, 15
13634 word embedding, 67
13635 word embeddings, 61, 148, 334, 335
13636 word representations, 334
13637 word sense disambiguation, 85
13638 word senses, 85
13639 word tokens, 88
13640 WordNet, 20
13641 world model, 288
- 13642 yield (context-free grammars), 217
13643 zero-one loss, 44