

1

Natural Language Processing

2

Jacob Eisenstein

3

September 15, 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	46
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	63
41	3.1 Feedforward neural networks	64
42	3.2 Designing neural networks	66
43	3.2.1 Activation functions	66
44	3.2.2 Network structure	67
45	3.2.3 Outputs and loss functions	68
46	3.2.4 Inputs and lookup layers	69
47	3.3 Learning neural networks	69
48	3.3.1 Backpropagation	71
49	3.3.2 Regularization and dropout	73
50	3.3.3 *Learning theory	74
51	3.3.4 Tricks	75
52	3.4 Convolutional neural networks	77
53	4 Linguistic applications of classification	85
54	4.1 Sentiment and opinion analysis	85
55	4.1.1 Related problems	87
56	4.1.2 Alternative approaches to sentiment analysis	88
57	4.2 Word sense disambiguation	89
58	4.2.1 How many word senses?	90
59	4.2.2 Word sense disambiguation as classification	91
60	4.3 Design decisions for text classification	92
61	4.3.1 What is a word?	92
62	4.3.2 How many words?	95
63	4.3.3 Count or binary?	96
64	4.4 Evaluating classifiers	97
65	4.4.1 Precision, recall, and <i>F</i> -MEASURE	97
66	4.4.2 Threshold-free metrics	99
67	4.4.3 Classifier comparison and statistical significance	100
68	4.4.4 *Multiple comparisons	103
69	4.5 Building datasets	104
70	4.5.1 Metadata as labels	104

71	4.5.2 Labeling data	104
72	5 Learning without supervision	111
73	5.1 Unsupervised learning	111
74	5.1.1 K -means clustering	112
75	5.1.2 Expectation-Maximization (EM)	114
76	5.1.3 EM as an optimization algorithm	118
77	5.1.4 How many clusters?	119
78	5.2 Applications of expectation-maximization	120
79	5.2.1 Word sense induction	120
80	5.2.2 Semi-supervised learning	121
81	5.2.3 Multi-component modeling	122
82	5.3 Semi-supervised learning	123
83	5.3.1 Multi-view learning	124
84	5.3.2 Graph-based algorithms	125
85	5.4 Domain adaptation	126
86	5.4.1 Supervised domain adaptation	127
87	5.4.2 Unsupervised domain adaptation	128
88	5.5 *Other approaches to learning with latent variables	130
89	5.5.1 Sampling	130
90	5.5.2 Spectral learning	132
91	II Sequences and trees	139
92	6 Language models	141
93	6.1 N -gram language models	142
94	6.2 Smoothing and discounting	145
95	6.2.1 Smoothing	145
96	6.2.2 Discounting and backoff	146
97	6.2.3 *Interpolation	147
98	6.2.4 *Kneser-Ney smoothing	149
99	6.3 Recurrent neural network language models	150
100	6.3.1 Backpropagation through time	152
101	6.3.2 Hyperparameters	153
102	6.3.3 Gated recurrent neural networks	153
103	6.4 Evaluating language models	155
104	6.4.1 Held-out likelihood	155
105	6.4.2 Perplexity	156
106	6.5 Out-of-vocabulary words	157

107	7 Sequence labeling	161
108	7.1 Sequence labeling as classification	161
109	7.2 Sequence labeling as structure prediction	163
110	7.3 The Viterbi algorithm	165
111	7.3.1 Example	168
112	7.3.2 Higher-order features	169
113	7.4 Hidden Markov Models	169
114	7.4.1 Estimation	171
115	7.4.2 Inference	171
116	7.5 Discriminative sequence labeling with features	173
117	7.5.1 Structured perceptron	176
118	7.5.2 Structured support vector machines	176
119	7.5.3 Conditional random fields	178
120	7.6 Neural sequence labeling	183
121	7.6.1 Recurrent neural networks	183
122	7.6.2 Character-level models	185
123	7.6.3 Convolutional Neural Networks for Sequence Labeling	186
124	7.7 *Unsupervised sequence labeling	186
125	7.7.1 Linear dynamical systems	188
126	7.7.2 Alternative unsupervised learning methods	188
127	7.7.3 Semiring notation and the generalized viterbi algorithm	188
128	8 Applications of sequence labeling	191
129	8.1 Part-of-speech tagging	191
130	8.1.1 Parts-of-Speech	192
131	8.1.2 Accurate part-of-speech tagging	196
132	8.2 Morphosyntactic Attributes	198
133	8.3 Named Entity Recognition	199
134	8.4 Tokenization	201
135	8.5 Code switching	202
136	8.6 Dialogue acts	203
137	9 Formal language theory	207
138	9.1 Regular languages	208
139	9.1.1 Finite state acceptors	209
140	9.1.2 Morphology as a regular language	210
141	9.1.3 Weighted finite state acceptors	212
142	9.1.4 Finite state transducers	217
143	9.1.5 *Learning weighted finite state automata	222
144	9.2 Context-free languages	223
145	9.2.1 Context-free grammars	224

146	9.2.2 Natural language syntax as a context-free language	227
147	9.2.3 A phrase-structure grammar for English	229
148	9.2.4 Grammatical ambiguity	234
149	9.3 *Mildly context-sensitive languages	234
150	9.3.1 Context-sensitive phenomena in natural language	235
151	9.3.2 Combinatory categorial grammar	236
152	10 Context-free parsing	241
153	10.1 Deterministic bottom-up parsing	242
154	10.1.1 Recovering the parse tree	244
155	10.1.2 Non-binary productions	244
156	10.1.3 Complexity	245
157	10.2 Ambiguity	245
158	10.2.1 Parser evaluation	246
159	10.2.2 Local solutions	247
160	10.3 Weighted Context-Free Grammars	248
161	10.3.1 Parsing with weighted context-free grammars	249
162	10.3.2 Probabilistic context-free grammars	251
163	10.3.3 *Semiring weighted context-free grammars	253
164	10.4 Learning weighted context-free grammars	253
165	10.4.1 Probabilistic context-free grammars	254
166	10.4.2 Feature-based parsing	254
167	10.4.3 *Conditional random field parsing	255
168	10.4.4 Neural context-free grammars	257
169	10.5 Grammar refinement	258
170	10.5.1 Parent annotations and other tree transformations	259
171	10.5.2 Lexicalized context-free grammars	260
172	10.5.3 *Refinement grammars	264
173	10.6 Beyond context-free parsing	265
174	10.6.1 Reranking	265
175	10.6.2 Transition-based parsing	266
176	11 Dependency parsing	271
177	11.1 Dependency grammar	271
178	11.1.1 Heads and dependents	272
179	11.1.2 Labeled dependencies	273
180	11.1.3 Dependency subtrees and constituents	274
181	11.2 Graph-based dependency parsing	276
182	11.2.1 Graph-based parsing algorithms	278
183	11.2.2 Computing scores for dependency arcs	279
184	11.2.3 Learning	281

185	11.3 Transition-based dependency parsing	282
186	11.3.1 Transition systems for dependency parsing	283
187	11.3.2 Scoring functions for transition-based parsers	287
188	11.3.3 Learning to parse	288
189	11.4 Applications	291
190	III Meaning	297
191	12 Logical semantics	299
192	12.1 Meaning and denotation	300
193	12.2 Logical representations of meaning	301
194	12.2.1 Propositional logic	301
195	12.2.2 First-order logic	302
196	12.3 Semantic parsing and the lambda calculus	306
197	12.3.1 The lambda calculus	307
198	12.3.2 Quantification	309
199	12.4 Learning semantic parsers	311
200	12.4.1 Learning from derivations	312
201	12.4.2 Learning from logical forms	314
202	12.4.3 Learning from denotations	315
203	13 Predicate-argument semantics	321
204	13.1 Semantic roles	323
205	13.1.1 VerbNet	324
206	13.1.2 Proto-roles and PropBank	325
207	13.1.3 FrameNet	326
208	13.2 Semantic role labeling	328
209	13.2.1 Semantic role labeling as classification	328
210	13.2.2 Semantic role labeling as constrained optimization	331
211	13.2.3 Neural semantic role labeling	333
212	13.3 Abstract Meaning Representation	334
213	13.3.1 AMR Parsing	337
214	13.4 Applications of Predicate-Argument Semantics	338
215	14 Distributional and distributed semantics	345
216	14.1 The distributional hypothesis	345
217	14.2 Design decisions for word representations	347
218	14.2.1 Representation	347
219	14.2.2 Context	348
220	14.2.3 Estimation	349

221	14.3 Latent semantic analysis	350
222	14.4 Brown clusters	351
223	14.5 Neural word embeddings	355
224	14.5.1 Continuous bag-of-words (CBOW)	355
225	14.5.2 Skipgrams	356
226	14.5.3 Computational complexity	356
227	14.5.4 Word embeddings as matrix factorization	358
228	14.6 Evaluating word embeddings	359
229	14.6.1 Intrinsic evaluations	359
230	14.6.2 Extrinsic evaluations	360
231	14.7 Distributed representations beyond distributional statistics	361
232	14.7.1 Word-internal structure	362
233	14.7.2 Lexical semantic resources	364
234	14.8 Distributed representations of multiword units	364
235	14.8.1 Purely distributional methods	364
236	14.8.2 Distributional-compositional hybrids	365
237	14.8.3 Supervised compositional methods	366
238	14.8.4 Hybrid distributed-symbolic representations	367
239	15 Reference Resolution	371
240	15.1 Forms of referring expressions	372
241	15.1.1 Pronouns	372
242	15.1.2 Proper Nouns	377
243	15.1.3 Nominals	378
244	15.2 Algorithms for coreference resolution	378
245	15.2.1 Mention-pair models	379
246	15.2.2 Mention-ranking models	380
247	15.2.3 Transitive closure in mention-based models	381
248	15.2.4 Entity-based models	382
249	15.3 Representations for coreference resolution	387
250	15.3.1 Features	388
251	15.3.2 Distributed representations of mentions and entities	390
252	15.4 Evaluating coreference resolution	393
253	16 Discourse	399
254	16.1 Segments	399
255	16.1.1 Topic segmentation	400
256	16.1.2 Functional segmentation	401
257	16.2 Entities and reference	401
258	16.2.1 Centering theory	402
259	16.2.2 The entity grid	403

260	16.2.3 *Formal semantics beyond the sentence level	404
261	16.3 Relations	404
262	16.3.1 Shallow discourse relations	405
263	16.3.2 Hierarchical discourse relations	408
264	16.3.3 Argumentation	412
265	16.3.4 Applications of discourse relations	413
266	IV Applications	421
267	17 Information extraction	423
268	17.1 Entities	425
269	17.1.1 Entity linking by learning to rank	426
270	17.1.2 Collective entity linking	428
271	17.1.3 *Pairwise ranking loss functions	429
272	17.2 Relations	431
273	17.2.1 Pattern-based relation extraction	432
274	17.2.2 Relation extraction as a classification task	433
275	17.2.3 Knowledge base population	436
276	17.2.4 Open information extraction	440
277	17.3 Events	441
278	17.4 Hedges, denials, and hypotheticals	442
279	17.5 Question answering and machine reading	444
280	17.5.1 Formal semantics	444
281	17.5.2 Machine reading	445
282	18 Machine translation	453
283	18.1 Machine translation as a task	453
284	18.1.1 Evaluating translations	455
285	18.1.2 Data	457
286	18.2 Statistical machine translation	458
287	18.2.1 Statistical translation modeling	459
288	18.2.2 Estimation	460
289	18.2.3 Phrase-based translation	461
290	18.2.4 *Syntax-based translation	463
291	18.3 Neural machine translation	464
292	18.3.1 Neural attention	466
293	18.3.2 *Neural machine translation without recurrence	468
294	18.3.3 Out-of-vocabulary words	469
295	18.4 Decoding	471
296	18.5 Training towards the evaluation metric	473

297	19 Text generation	479
298	19.1 Data-to-text generation	479
299	19.1.1 Latent data-to-text alignment	481
300	19.1.2 Neural data-to-text generation	482
301	19.2 Text-to-text generation	486
302	19.2.1 Neural abstractive summarization	486
303	19.2.2 Sentence fusion for multi-document summarization	488
304	19.3 Dialogue	489
305	19.3.1 Finite-state and agenda-based dialogue systems	489
306	19.3.2 Markov decision processes	490
307	19.3.3 Neural chatbots	492
308	A Probability	497
309	A.1 Probabilities of event combinations	497
310	A.1.1 Probabilities of disjoint events	498
311	A.1.2 Law of total probability	499
312	A.2 Conditional probability and Bayes' rule	499
313	A.3 Independence	501
314	A.4 Random variables	502
315	A.5 Expectations	503
316	A.6 Modeling and estimation	504
317	B Numerical optimization	507
318	B.1 Gradient descent	508
319	B.2 Constrained optimization	508
320	B.3 Example: Passive-aggressive online learning	509
321	Bibliography	511

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 x and parameters θ . 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

$x^{(i)}$	a column vector of feature counts for instance i , often word counts
$x_{j:k}$	elements j through k (inclusive) of a vector x
$[x; y]$	vertical concatenation of two column vectors
$[x, y]$	horizontal concatenation of two column vectors
e_n	a “one-hot” vector with a value of 1 at position n , and zero everywhere else
θ^\top	the transpose of a column vector θ
$\theta \cdot x^{(i)}$	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}
$\text{Diag}(x)$	a matrix with 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**¹ (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.²

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

¹Throughout the text, **boldface** will be used to indicate keywords that appear in the index.

²Throughout the text, this notation will be used to introduce linguistic examples.

386 phrase, which is conjoined with the prepositional phrase *of the bigness of a human*
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.³

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

³This view is 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 despite its many practical applications, language is perhaps “number 300” in the priority list for artificial intelligence research, and that it would be a great achievement if AI could attain the capabilities of an orangutan, which do not include language (<http://www.abigailsee.com/2018/02/21/deep-learning-structure-and-innate-priors.html>).

415 subdisciplines such as natural language processing, machine learning, and computer vi-
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. Large
433 datasets of unlabeled text are a natural application for parallelization techniques like
434 MapReduce (Dean and Ghemawat, 2008; Lin and Dyer, 2010); high-volume data sources
435 such as social media are a natural application for approximate streaming and sketching
436 techniques (Goyal et al., 2009). When deep neural networks are implemented in pro-
437 duction systems, it is possible to eke out speed gains using techniques such as reduced-
438 precision arithmetic (Wu et al., 2016). Many classical natural language processing algo-
439 rithms are not naturally suited to graphics processing unit (GPU) parallelization, suggest-
440 ing directions for further research at the intersection of natural language processing and
441 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 models**, 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.⁴ 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 (morphology)**, so that
 501 a learning system can more easily generalize across related terms such as *whale*, *whales*,
 502 *whalers*, and *whaling*. This is particularly important in the many languages that exceed
 503 English in the complexity of the system of affixes that can attach to words. Such features
 504 could be obtained from a hand-crafted resource, like a dictionary that maps each word to
 505 a single root form. Alternatively, features can be obtained from the output of a general-
 506 purpose language processing system, such as a parser or part-of-speech tagger, which
 507 may itself 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;⁵ 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

⁴The *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).

⁵Ali 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,⁶

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

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 (machine learning)**), which maps from
 531 the set $\mathcal{X} \times \mathcal{Y}$ to 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**.

⁶Throughout 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.⁷ 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.

⁷<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 colleagues and students read early drafts of chapters in their areas of expertise, in-
733 cluding Yoav Artzi, Kevin Duh, Heng Ji, Jessy Li, Brendan O'Connor, Yuval Pinter, Nathan
734 Schneider, Pamela Shapiro, Noah A. Smith, Sandeep Soni, and Luke Zettlemoyer. I would
735 also like to thank the following people for helpful discussions: Kevin Murphy, Shawn
736 Ling Ramirez, William Yang Wang, and Bonnie Webber. In addition, students, colleagues,
737 friends, and family found mistakes in early drafts: Parminder Bhatia, Kimberly Caras,
738 Jiahao Cai, Justin Chen, Murtaza Dhuliawala, Barbara Eisenstein, Luiz C. F. Ribeiro, Chris
739 Gu, Joshua Killingsworth, Jonathan May, Taha Merghani, Gus Monod, Raghavendra Mu-
740 rali, Nidish Nair, Brendan O'Connor, Brandon Peck, Yuval Pinter, Nathan Schneider, Jian-
741 hao Shen, Zhewei Sun, Rubin Tsui, Ashwin Cunnappakkam Vinjimir, Denny Vrandečić,
742 Clay Washington, Ishan Waykul, Xavier Yao, and Yuyu Zhang. Clay Washington pilot
743 tested several of the programming exercises. Special thanks to the many students in Geor-
744 gia Tech's CS 4650 and 7650 who suffered through early versions of the text.

745

Part I

746

Learning

747

Chapter 2

748

Linear text classification

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

754 To perform this task, the first question is how to represent each document. A common
755 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
756 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
757 in the vocabulary.

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

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

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

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

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

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

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

778 This function returns the count of the word *freeee* if the label is SPAM, and it returns zero
 779 otherwise. The corresponding weight θ_j then scores the compatibility of the word *freeee*
 780 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,

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

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

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

781 where $\underbrace{[0; 0; \dots; 0]}_{(K-1) \times V}$ is a column vector of $(K - 1) \times V$ zeros, and the semicolon indicates
 782 vertical concatenation. This arrangement is shown in Figure 2.1; the notation may seem
 783 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} ,

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

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

784 This inner product notation gives a clean separation between the *data* (x and y) and the
 785 *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 $y \in \mathcal{Y}$.

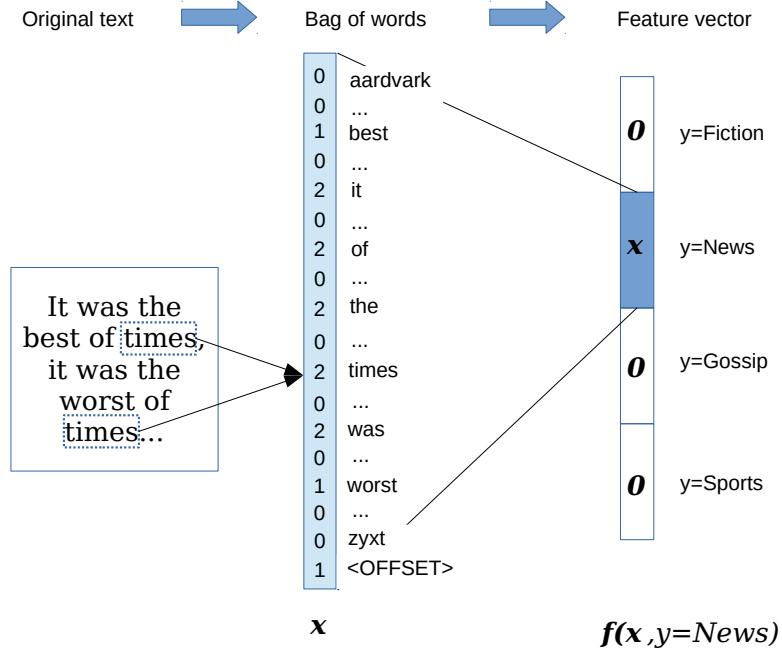


Figure 2.1: The bag-of-words and feature vector representations, for a hypothetical text classification task.

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

788 It is common to add an **offset feature** at the end of the vector of word counts \mathbf{x} , which
 789 is always 1. We then have to also add an extra zero to each of the zero vectors, to make the
 790 vector lengths match. This gives the entire feature vector $\mathbf{f}(\mathbf{x}, y)$ a length of $(V + 1) \times K$.
 791 The weight associated with this offset feature can be thought of as a bias for or against
 792 each label. For example, if we expect most documents to be spam, then the weight for
 793 the offset feature for $y = \text{SPAM}$ should be larger than the weight for the offset feature for
 794 $y = \text{HAM}$.

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

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

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

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

805 2.1 Naïve Bayes

806 The **joint probability** of a bag of words \mathbf{x} and its true label y is written $p(\mathbf{x}, y)$. Suppose
 807 we have a dataset of N labeled instances, $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$, which we assume are **independ-**
 808 **ent and identically distributed (IID)** (see § A.3). Then the joint probability of the entire
 809 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 $\mathbf{x}^{(i)} | y^{(i)} \sim \text{Multinomial}(\boldsymbol{\phi}_{y^{(i)}})$.

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

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

- 819 • The first line of this generative model encodes the assumption that the instances are
 820 mutually independent: neither the label nor the text of document i affects the label
 821 or text of document j .⁶ Furthermore, the instances are identically distributed: the
 822 distributions over the label $y^{(i)}$ and the text $\mathbf{x}^{(i)}$ (conditioned on $y^{(i)}$) are the same
 823 for all instances i .
- 824 • The second line of the generative model states that the random variable $y^{(i)}$ is drawn
 825 from a categorical distribution with parameter $\boldsymbol{\mu}$. Categorical distributions are like
 826 weighted dice: the vector $\boldsymbol{\mu} = [\mu_1, \mu_2, \dots, \mu_K]^\top$ gives the probabilities of each la-
 827 bel, so that the probability of drawing label y is equal to μ_y . For example, if $\mathcal{Y} =$
 828 $\{\text{POSITIVE}, \text{NEGATIVE}, \text{NEUTRAL}\}$, we might have $\boldsymbol{\mu} = [0.1, 0.7, 0.2]^\top$. We require
 829 $\sum_{y \in \mathcal{Y}} \mu_y = 1$ and $\mu_y \geq 0, \forall y \in \mathcal{Y}$.⁷
- 830 • The third line describes how the bag-of-words counts $\mathbf{x}^{(i)}$ are generated. By writing
 831 $\mathbf{x}^{(i)} | y^{(i)}$, 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 .

832 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:

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

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

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

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

845 2.1.1 Types and tokens

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

⁸Technically, a multinomial distribution requires a second parameter, the total number of word counts in 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  $w_m^{(i)} | y^{(i)} \sim \text{Categorical}(\boldsymbol{\phi}_{y^{(i)}})$ .

```

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

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

870 **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

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

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

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

878 2.1.3 Estimation

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

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

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

883 Equation 2.21 defines the **relative frequency estimate** for ϕ . It can be justified as a
 884 **maximum likelihood estimate**: the estimate that maximizes the probability $p(\mathbf{x}^{(1:N)}, y^{(1:N)}; \boldsymbol{\theta})$.
 885 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 .

Maximum-likelihood estimation chooses ϕ to maximize the log-likelihood \mathcal{L} . However, the solution must obey the following 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 to the objective (see Appendix B for more details). To solve for each θ_y , we maximize the Lagrangian,

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

Differentiating with respect to the parameter $\phi_{y,j}$ yields,

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

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**—see **indicator function**, also sometimes called an **indicator function**—see **delta 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,

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

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

893 2.1.4 Smoothing and MAP estimation

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

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

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

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

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

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

914 2.1.5 Setting hyperparameters

915 How should we choose the best value of hyperparameters like α ? Maximum likelihood
 916 will not work: the maximum likelihood estimate of α on the training set will always be
 917 $\alpha = 0$. In many cases, what we really want is **accuracy**: the number of correct predictions,
 918 divided by the total number of predictions. (Other measures of classification performance
 919 are discussed in § 4.4.) As we will see, it is hard to optimize for accuracy directly. But for
 920 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.

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

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

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

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

947 2.2 Discriminative learning

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

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

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

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

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

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

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

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

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

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

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

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

2.2.1 Perceptron

In Naïve Bayes, the weights can be interpreted as parameters of a probabilistic model. But 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  $\boldsymbol{\theta}^{(t)}$ 

```

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

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

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

989 **Definition 1** (Linear separability). *The dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$ is linearly separable iff
990 (if and only if) there exists some weight vector $\boldsymbol{\theta}$ and some margin ρ such that for every instance
991 $(\mathbf{x}^{(i)}, y^{(i)})$, 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
992 at least ρ greater than inner product of $\boldsymbol{\theta}$ and the feature function for every other possible label,
993 $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y')$.*

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

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

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

998 How useful is this proof? Minsky and Papert (1969) famously proved that the simple
 999 logical function of *exclusive-or* is not separable, and that a perceptron is therefore inca-
 1000 pable of learning this function. But this is not just an issue for the perceptron: any linear
 1001 classification algorithm, including Naïve Bayes, will fail on this task. In natural language
 1002 classification problems usually involve high dimensional feature spaces, with thousands
 1003 or millions of features. For these problems, it is very likely that the training data is indeed
 1004 separable. And even if the data is not separable, it is still possible to place an upper bound
 1005 on the number of errors that the perceptron algorithm will make (Freund and Schapire,
 1006 1999).

1007 2.2.2 Averaged perceptron

1008 The perceptron iterates over the data repeatedly — until “tired”, as described in Algo-
 1009 rithm 3. If the data is linearly separable, the perceptron will eventually find a separator,
 1010 and we can stop once all training instances are classified correctly. But if the data is not
 1011 linearly separable, the perceptron can *thrash* between two or more weight settings, never
 1012 converging. In this case, how do we know that we can stop training, and how should
 1013 we choose the final weights? An effective practical solution is to *average* the perceptron
 1014 weights across all iterations.

1015 This procedure is shown in Algorithm 4. The learning algorithm is nearly identical,
 1016 but we also maintain a vector of the sum of the weights, \mathbf{m} . At the end of the learning
 1017 procedure, we divide this sum by the total number of updates t , to compute the average
 1018 weights, $\bar{\theta}$. These average weights are then used for prediction. In the algorithm sketch,
 1019 the average is computed from a running sum, $\mathbf{m} \leftarrow \mathbf{m} + \theta$. However, this is inefficient,
 1020 because it requires $|\theta|$ operations to update the running sum. When $f(\mathbf{x}, y)$ is sparse,
 1021 $|\theta| \gg |f(\mathbf{x}, y)|$ for any individual (\mathbf{x}, y) . This means that computing the running sum will
 1022 be much more expensive than computing of the update to θ itself, which requires only
 1023 $2 \times |f(\mathbf{x}, y)|$ operations. One of the exercises is to sketch a more efficient algorithm for
 1024 computing the averaged weights.

1025 Even if the data is not separable, the averaged weights will eventually converge. One
 1026 possible stopping criterion is to check the difference between the average weight vectors
 1027 after each pass through the data: if the norm of the difference falls below some predefined
 1028 threshold, we can stop training. Another stopping criterion is to hold out some data,
 1029 and to measure the predictive accuracy on this heldout data. When the accuracy on the
 1030 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 **machine 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  $\bar{\boldsymbol{\theta}}$ 

```

1031 set. At this point, it is probably best to stop; this stopping criterion is known as **early**
 1032 **stopping**.

1033 **Generalization** is the ability to make good predictions on instances that are not in
 1034 the training data. Averaging can be proven to improve generalization, by computing an
 1035 upper bound on the generalization error (Freund and Schapire, 1999; Collins, 2002).

1036 **2.3 Loss functions and large-margin classification**

1037 Naïve Bayes chooses the weights $\boldsymbol{\theta}$ by maximizing the joint log-likelihood $\log p(\mathbf{x}^{(1:N)}, \mathbf{y}^{(1:N)})$.
 1038 By convention, optimization problems are generally formulated as minimization of a **loss**
 1039 **function**. The input to a loss function is the vector of weights $\boldsymbol{\theta}$, and the output is a non-
 1040 negative scalar, measuring the performance of the classifier on a training instance. The
 1041 loss $\ell(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)})$ is then a measure of the performance of the weights $\boldsymbol{\theta}$ on the instance
 1042 $(\mathbf{x}^{(i)}, y^{(i)})$. The goal of learning is to minimize the sum of the losses across all instances in
 1043 the training set.

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

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

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

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

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

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

1044 The problem of minimizing ℓ_{NB} is thus identical to the problem of maximum-likelihood
 1045 estimation.

1046 Loss functions provide a general framework for comparing machine learning objectives.
 1047 For example, an alternative loss function is the **zero-one loss**,

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

1048 The zero-one loss is zero if the instance is correctly classified, and one otherwise. The
 1049 sum of zero-one losses is proportional to the error rate of the classifier on the training
 1050 data. Since a low error rate is often the ultimate goal of classification, this may seem
 1051 ideal. But the zero-one loss has several problems. One is that it is **non-convex**,¹³ which
 1052 means that there is no guarantee that gradient-based optimization will be effective. A
 1053 more serious problem is that the derivatives are useless: the partial derivative with respect
 1054 to any parameter is zero everywhere, except at the points where $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y) = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, \hat{y})$
 1055 for some \hat{y} . At those points, the loss is discontinuous, and the derivative is undefined.

1056 The perceptron optimizes the following loss function:

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

1057 When $\hat{y} = y^{(i)}$, the loss is zero; otherwise, it increases linearly with the gap between the
 1058 score for the predicted label \hat{y} and the score for the true label $y^{(i)}$. Plotting this loss against
 1059 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
 1060 **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.

1061 To see why this is the loss function optimized by the perceptron, take the derivative
 1062 with respect to θ ,

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

1063 At each instance perceptron algorithm takes a step of magnitude one in the opposite direction
 1064 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,
 1065 this is an example of the optimization algorithm **stochastic gradient descent**, applied to
 1066 the objective in Equation 2.41.

1067 **Breaking ties with subgradient descent** Careful readers will notice the tacit assumption
 1068 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
 1069 that maximize this function? Consider binary classification: if the maximizer is $y^{(i)}$, then
 1070 the gradient is zero, and so is the perceptron update; if the maximizer is $\hat{y} \neq y^{(i)}$, then the
 1071 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
 1072 loss is not **smooth**, because the first derivative has a discontinuity at the hinge point,
 1073 where the score for the true label $y^{(i)}$ is equal to the score for some other label \hat{y} . At this
 1074 point, there is no unique gradient; rather, there is a set of **subgradients**. A vector v is a
 1075 subgradient of the function g at u_0 iff $g(u) - g(u_0) \geq v \cdot (u - u_0)$ for all u . Graphically,
 1076 this defines the set of hyperplanes that include $g(u_0)$ and do not intersect g at any other
 1077 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
 1078 approach from the right, the gradient is 0. At the hinge point, the subgradients include all
 1079 vectors that are bounded by these two extremes. In subgradient descent, *any* subgradient
 1080 can be used (Bertsekas, 2012). Since both 0 and $\mathbf{f}(\mathbf{x}, \hat{y}) - \mathbf{f}(\mathbf{x}, y)$ are subgradients at the
 1081 hinge point, either one can be used in the perceptron update.

1082 **Perceptron versus Naïve Bayes** The perceptron loss function has some pros and cons
 1083 with respect to the negative log-likelihood loss implied by Naïve Bayes.

- 1084 • Both ℓ_{NB} and $\ell_{\text{PERCEPTRON}}$ are convex, making them relatively easy to optimize. How-
 1085 ever, ℓ_{NB} can be optimized in closed form, while $\ell_{\text{PERCEPTRON}}$ requires iterating over
 1086 the dataset multiple times.
- 1087 • ℓ_{NB} can suffer **infinite** loss on a single example, since the logarithm of zero probabil-
 1088 ity is negative infinity. Naïve Bayes will therefore overemphasize some examples,
 1089 and underemphasize others.
- 1090 • $\ell_{\text{PERCEPTRON}}$ treats all correct answers equally. Even if θ only gives the correct answer
 1091 by a tiny margin, the loss is still zero.

1092 **2.3.1 Large margin classification**

1093 This last comment suggests a potential problem with the perceptron. Suppose a test ex-
 1094 ample is very close to a training example, but not identical. If the classifier only gets the
 1095 correct answer on the training example by a small margin, then it may get the test instance
 1096 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,

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

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

1097 where $(x)_+ = \max(0, x)$. The loss is zero if there is a margin of at least 1 between the
 1098 score for the true label and the best-scoring alternative \hat{y} . This is almost identical to the
 1099 perceptron loss, but the hinge point is shifted to the right, as shown in Figure 2.2. The
 1100 margin loss is a convex upper bound on the zero-one loss.

1101 **2.3.2 Support vector machines**

If a dataset is linearly separable, then there is some hyperplane $\boldsymbol{\theta}$ 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 $(\mathbf{x}^{(i)}, y^{(i)})$, the geometric distance to the separating hyperplane is given by $\frac{\gamma(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)})}{\|\boldsymbol{\theta}\|_2}$,

where the denominator is the norm of the weights, $\|\boldsymbol{\theta}\|_2 = \sqrt{\sum_j \theta_j^2}$. The geometric distance is sometimes called the **geometric margin**, in contrast to the **functional margin** $\gamma(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)})$. 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 $\|\boldsymbol{\theta}\|_2$ is also large, then a small change in $\mathbf{x}^{(i)}$ 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:

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

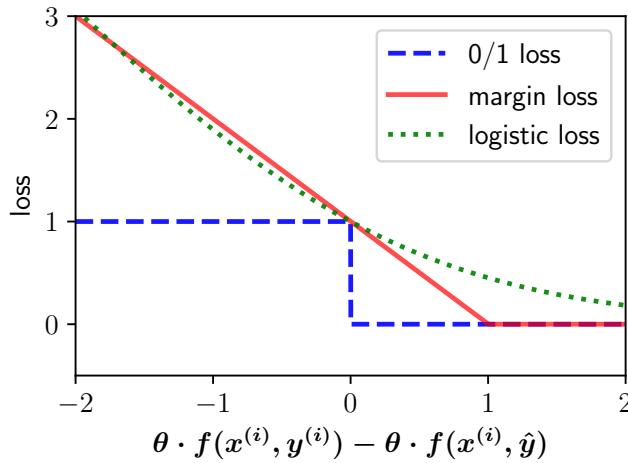


Figure 2.2: Margin, zero-one, and logistic loss functions.

1102 This is a **constrained optimization** problem, where the second line describes constraints
 1103 on the space of possible solutions θ . In this case, the constraint is that the functional
 1104 margin always be at least one, and the objective is that the minimum geometric margin
 1105 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,

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

1106 This optimization problem is a **quadratic program**: the objective is a quadratic func-
 1107 tion of the parameters, and the constraints are all linear inequalities. The resulting clas-
 1108 sifier is better known as the **support vector machine**. The name derives from one of the
 1109 solutions, which is to incorporate the constraints through Lagrange multipliers $\alpha_i \geq 0, i =$
 1110 $1, 2, \dots, N$. The instances for which $\alpha_i > 0$ are the **support vectors**; other instances are
 1111 irrelevant to the classification boundary.

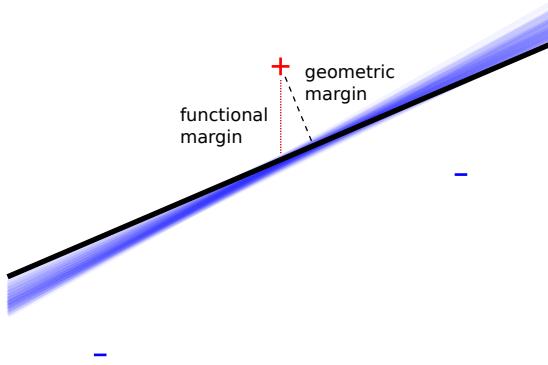


Figure 2.3: Functional and geometric margins for a binary classification problem. All separators that satisfy the margin constraint are shown. The separator with the largest geometric margin is shown in bold.

1112 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:

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

1113 The hyperparameter C controls the tradeoff between violations of the margin con-
 1114 straint and the preference for a low norm of θ . As $C \rightarrow \infty$, slack is infinitely expensive,
 1115 and there is only a solution if the data is separable. As $C \rightarrow 0$, slack becomes free, and
 1116 there is a trivial solution at $\theta = 0$. Thus, C plays a similar role to the smoothing parame-
 1117 ter in Naïve Bayes (§ 2.1.4), trading off between a close fit to the training data and better
 1118 generalization. Like the smoothing parameter of Naïve Bayes, C must be set by the user,
 1119 typically by maximizing performance on a heldout development set.

1120 To solve the constrained optimization problem defined in Equation 2.48, we can first

1121 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,

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

1122 where each ξ_i has been substituted by the right-hand side of Equation 2.49, and the factor
 1123 of C on the slack variables has been replaced by an equivalent factor of $\lambda = \frac{1}{C}$ on the
 1124 norm of the weights.

1125 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,

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

1126 This objective maximizes over all $y \in \mathcal{Y}$, in search of labels that are both *strong*, as mea-
 1127 sured by $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y)$, and *wrong*, as measured by $c(y^{(i)}, y)$. This maximization is known
 1128 as **cost-augmented decoding**, because it augments the maximization objective to favor
 1129 high-cost predictions. If the highest-scoring label is $y = y^{(i)}$, then the margin constraint is
 1130 satisfied, and the loss for this instance is zero. Cost-augmentation is only for learning: it
 1131 is not applied when making predictions on unseen data.

Differentiating Equation 2.52 with respect to the weights gives,

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

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

1132 where L_{SVM} refers to minimization objective in Equation 2.52. This gradient is very similar
 1133 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.

1134 weights towards 0. The other difference is the cost $c(y^{(i)}, y)$, which is added to $\theta \cdot \mathbf{f}(\mathbf{x}, y)$
 1135 when choosing \hat{y} during training. This term derives from the margin constraint: large
 1136 margin classifiers learn not only from instances that are incorrectly classified, but also
 1137 from instances for which the correct classification decision was not sufficiently confident.

1138 2.4 Logistic regression

1139 Thus far, we have seen two broad classes of learning algorithms. Naïve Bayes is a prob-
 1140 abilistic method, where learning is equivalent to estimating a joint probability distribu-
 1141 tion. The perceptron and support vector machine are discriminative, error-driven algo-
 1142 rithms: the learning objective is closely related to the number of errors on the training
 1143 data. Probabilistic and error-driven approaches each have advantages: probability makes
 1144 it possible to quantify uncertainty about the predicted labels, but the probability model of
 1145 Naïve Bayes makes unrealistic independence assumptions that limit the features that can
 1146 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**,

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

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

1147 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.

1148 the sum, we have the (additive inverse of the) **logistic loss**,

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

1149 The logistic loss is shown in Figure 2.2. A key difference from the zero-one and hinge
 1150 losses is that logistic loss is never zero. This means that the objective function can always
 1151 be improved by assigning higher confidence to the correct label.

1152 2.4.1 Regularization

1153 As with the support vector machine, better generalization can be obtained by penalizing
 1154 the norm of $\boldsymbol{\theta}$. This is done by adding a term of $\frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2$ to the minimization objective.
 1155 This is called L_2 regularization, because $\|\boldsymbol{\theta}\|_2^2$ is the squared L_2 norm of the vector $\boldsymbol{\theta}$.
 1156 Regularization forces the estimator to trade off performance on the training data against
 1157 the norm of the weights, and this can help to prevent overfitting. Consider what would
 1158 happen to the unregularized weight for a base feature j that is active in only one instance
 1159 $\mathbf{x}^{(i)}$: the conditional log-likelihood could always be improved by increasing the weight
 1160 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
 1161 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,

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

1162 so that the regularization weight λ is equal to the inverse variance of the prior, $\lambda = \frac{1}{\sigma^2}$.

1163 **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,

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

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

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

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

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

1164 The final step employs the definition of a conditional expectation (§ A.5). The gradient of
 1165 the logistic loss is equal to the difference between the expected counts under the current
 1166 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
 1167 vectors are equal for a single instance, there is nothing more to learn from it; when they
 1168 are equal in sum over the entire dataset, there is nothing more to learn from the dataset as
 1169 a whole. The gradient of the hinge loss is nearly identical, but it involves the features of
 1170 the predicted label under the current model, $\mathbf{f}(\mathbf{x}^{(i)}, \hat{y})$, rather than the expected features
 1171 $E_{Y|X}[\mathbf{f}(\mathbf{x}^{(i)}, y)]$ under the conditional distribution $p(y | \mathbf{x}; \boldsymbol{\theta})$.

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

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

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

1172 **2.5 Optimization**

1173 Each of the classification algorithms in this chapter can be viewed as an optimization
 1174 problem:

- 1175 • In Naïve Bayes, the objective is the joint likelihood $\log p(\mathbf{x}^{(1:N)}, \mathbf{y}^{(1:N)})$. Maximum
 1176 likelihood estimation yields a closed-form solution for $\boldsymbol{\theta}$.

- 1177 • In the support vector machine, the objective is the regularized margin loss,

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

1178 There is no closed-form solution, but the objective is convex. The perceptron algo-
1179 rithm minimizes a similar objective.

- 1180 • In logistic regression, the objective is the regularized negative log-likelihood,

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

1181 Again, there is no closed-form solution, but the objective is convex.

1182 These learning algorithms are distinguished by *what* is being optimized, rather than
1183 *how* the optimal weights are found. This decomposition is an essential feature of con-
1184 temporary machine learning. The domain expert's job is to design an objective function
1185 — or more generally, a **model** of the problem. If the model has certain characteristics,
1186 then generic optimization algorithms can be used to find the solution. In particular, if an
1187 objective function is differentiable, then gradient-based optimization can be employed;
1188 if it is also convex, then gradient-based optimization is guaranteed to find the globally
1189 optimal solution. The support vector machine and logistic regression have both of these
1190 properties, and so are amenable to generic **convex optimization** techniques (Boyd and
1191 Vandenberghe, 2004).

1192 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,

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

1193 where $\nabla_{\boldsymbol{\theta}} L$ is the gradient computed over the entire training set, and $\eta^{(t)}$ is the **learning**
1194 **rate** at iteration t . If the objective L is a convex function of $\boldsymbol{\theta}$, then this procedure is
1195 guaranteed to terminate at the global optimum, for appropriate schedule of learning rates,
1196 $\eta^{(t)}$.¹⁶

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

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

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

1197 In practice, gradient descent can be slow to converge, as the gradient can become
 1198 infinitesimally small. Faster convergence can be obtained by second-order Newton opti-
 1199 mization, which incorporates the inverse of the **Hessian matrix**,

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

1200 The size of the Hessian matrix is quadratic in the number of features. In the bag-of-words
 1201 representation, this is usually too big to store, let alone invert. **Quasi-Network optimiza-**
 1202 **tion** techniques maintain a low-rank approximation to the inverse of the Hessian matrix.
 1203 Such techniques usually converge more quickly than gradient descent, while remaining
 1204 computationally tractable even for large feature sets. A popular quasi-Newton algorithm
 1205 is L-BFGS (Liu and Nocedal, 1989), which is implemented in many scientific computing
 1206 environments, such as `scipy` and `Matlab`.

1207 For any gradient-based technique, the user must set the learning rates $\eta^{(t)}$. While con-
 1208 vergence proofs usually employ a decreasing learning rate, in practice, it is common to fix
 1209 $\eta^{(t)}$ to a small constant, like 10^{-3} . The specific constant can be chosen by experimentation,
 1210 although there is research on determining the learning rate automatically (Schaul et al.,
 1211 2013; Wu et al., 2018).

1212 2.5.2 Online optimization

1213 Batch optimization computes the objective on the entire training set before making an up-
 1214 date. This may be inefficient, because at early stages of training, a small number of train-
 1215 ing examples could point the learner in the correct direction. **Online learning** algorithms
 1216 make updates to the weights while iterating through the training data. The theoretical
 1217 basis for this approach is a stochastic approximation to the true objective function,

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

1218 where the instance $(\mathbf{x}^{(j)}, y^{(j)})$ is sampled at random from the full dataset.

1219 In **stochastic gradient descent**, the approximate gradient is computed by randomly
 1220 sampling a single instance, and an update is made immediately. This is similar to the
 1221 perceptron algorithm, which also updates the weights one instance at a time. In **mini-**
 1222 **batch** stochastic gradient descent, the gradient is computed over a small set of instances.
 1223 A typical approach is to set the minibatch size so that the entire batch fits in memory on a
 1224 graphics processing unit (GPU; Neubig et al., 2017). It is then possible to speed up learn-
 1225 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  $\boldsymbol{\theta}^{(t)}$ 

```

1226 Algorithm 5 offers a generalized view of gradient descent. In standard gradient de-
 1227 scent, the batcher returns a single batch with all the instances. In stochastic gradient de-
 1228 scent, it returns N batches with one instance each. In mini-batch settings, the batcher
 1229 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 learning rate, 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 learning rate; 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:

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

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

1230 where j iterates over features in $\mathbf{f}(\mathbf{x}, y)$.

1231 In most cases, the number of active features for any instance is much smaller than the
 1232 number of weights. If so, the computation cost of online optimization will be dominated
 1233 by the update from the regularization term, $\lambda \boldsymbol{\theta}$. The solution is to be “lazy”, updating
 1234 each θ_j only as it is used. To implement lazy updating, store an additional parameter τ_j ,
 1235 which is the iteration at which θ_j was last updated. If θ_j is needed at time t , the $t - \tau$

1236 regularization updates can be performed all at once. This strategy is described in detail
 1237 by Kummerfeld et al. (2015).

1238 2.6 *Additional topics in classification

1239 Throughout this text, advanced topics will be marked with an asterisk.

1240 2.6.1 Feature selection by regularization

1241 In logistic regression and large-margin classification, generalization can be improved by
 1242 regularizing the weights towards 0, using the L_2 norm. But rather than encouraging
 1243 weights to be small, it might be better for the model to be **sparse**: it should assign weights
 1244 of exactly zero to most features, and only assign non-zero weights to features that are
 1245 clearly necessary. This idea can be formalized by the L_0 norm, $L_0 = \|\theta\|_0 = \sum_j \delta(\theta_j \neq 0)$,
 1246 which applies a constant penalty for each non-zero weight. This norm can be thought
 1247 of as a form of **feature selection**: optimizing the L_0 -regularized conditional likelihood is
 1248 equivalent to trading off the log-likelihood against the number of active features. Reduc-
 1249 ing the number of active features is desirable because the resulting model will be fast,
 1250 low-memory, and should generalize well, since irrelevant features will be pruned away.
 1251 Unfortunately, the L_0 norm is non-convex and non-differentiable. Optimization under L_0
 1252 regularization is **NP-hard**, meaning that it can be solved efficiently only if P=NP (Ge et al.,
 1253 2011).

1254 A useful alternative is the L_1 norm, which is equal to the sum of the absolute values
 1255 of the weights, $\|\theta\|_1 = \sum_j |\theta_j|$. The L_1 norm is convex, and can be used as an approxima-
 1256 tion to L_0 (Tibshirani, 1996). Conveniently, the L_1 norm also performs feature selection,
 1257 by driving many of the coefficients to zero; it is therefore known as a **sparsity inducing**
 1258 **regularizer**. The L_1 norm does not have a gradient at $\theta_j = 0$, so we must instead optimize
 1259 the L_1 -regularized objective using **subgradient** methods. The associated stochastic sub-
 1260 gradient descent algorithms are only somewhat more complex than conventional SGD;
 1261 Sra et al. (2012) survey approaches for estimation under L_1 and other regularizers.

1262 Gao et al. (2007) compare L_1 and L_2 regularization on a suite of NLP problems, finding
 1263 that L_1 regularization generally gives similar accuracy to L_2 regularization, but that L_1
 1264 regularization produces models that are between ten and fifty times smaller, because more
 1265 than 90% of the feature weights are set to zero.

1266 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**,

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

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

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

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

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

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

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

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

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

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

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

1290 2.7 Summary of learning algorithms

1291 It is natural to ask which learning algorithm is best, but the answer depends on what
 1292 characteristics are important to the problem you are trying to solve.

1293 **Naïve Bayes** *Pros:* easy to implement; estimation is fast, requiring only a single pass over
 1294 the data; assigns probabilities to predicted labels; controls overfitting with smoothing
 1295 parameter. *Cons:* often has poor accuracy, especially with correlated features.

1296 **Perceptron** *Pros:* easy to implement; online; error-driven learning means that accuracy
 1297 is typically high, especially after averaging. *Cons:* not probabilistic; hard to know
 1298 when to stop learning; lack of margin can lead to overfitting.

1299 **Support vector machine** *Pros:* optimizes an error-based metric, usually resulting in high
 1300 accuracy; overfitting is controlled by a regularization parameter. *Cons:* not proba-
 1301 bilistic.

1302 **Logistic regression** *Pros:* error-driven and probabilistic; overfitting is controlled by a reg-
 1303 ularization parameter. *Cons:* batch learning requires black-box optimization; logistic
 1304 loss can “overtrain” on correctly labeled examples.

1305 One of the main distinctions is whether the learning algorithm offers a probability
 1306 over labels. This is useful in modular architectures, where the output of one classifier
 1307 is the input for some other system. In cases where probability is not necessary, the sup-
 1308 port vector machine is usually the right choice, since it is no more difficult to implement
 1309 than the perceptron, and is often more accurate. When probability is necessary, logistic
 1310 regression is usually more accurate than Naïve Bayes.

1311 Additional resources

1312 For more on classification, you can consult a textbook on machine learning (e.g., Mur-
 1313 phy, 2012), although the notation will differ slightly from what is typical in natural lan-
 1314 guage processing. Probabilistic methods are surveyed by Hastie et al. (2009), and Mohri
 1315 et al. (2012) emphasize theoretical considerations. Online learning is a rapidly moving
 1316 subfield of machine learning, and Bottou et al. (2016) describes progress through 2016.
 1317 Kummerfeld et al. (2015) empirically review several optimization algorithms for large-
 1318 margin learning. The python toolkit `scikit-learn` includes implementations of all of
 1319 the algorithms described in this chapter (Pedregosa et al., 2011).

1320 **Exercises**

1321 There will be exercises at the end of each chapter. In this chapter, the exercises are mostly
 1322 mathematical, matching the subject material. In other chapters, the exercises will em-
 1323 phasize linguistics or software development. In general, the exercises will be ordered by
 1324 increasing difficulty.

- 1325 1. Let \mathbf{x} be a bag-of-words vector such that $\sum_{j=1}^V x_j = 1$. Verify that the multinomial
 1326 probability $p_{\text{mult}}(\mathbf{x}; \boldsymbol{\phi})$, as defined in Equation 2.12, is identical to the probability of
 1327 the same document under a categorical distribution, $p_{\text{cat}}(\mathbf{w}; \boldsymbol{\phi})$.
2. Suppose you have a single feature x , with the following conditional distribution:

$$p(x | y) = \begin{cases} \alpha, & X = 0, Y = 0 \\ 1 - \alpha, & X = 1, Y = 0 \\ 1 - \beta, & X = 0, Y = 1 \\ \beta, & X = 1, Y = 1. \end{cases} \quad [2.81]$$

1328 Further suppose that the prior is uniform, $\Pr(Y = 0) = \Pr(Y = 1) = \frac{1}{2}$, and that
 1329 both $\alpha > \frac{1}{2}$ and $\beta > \frac{1}{2}$. Given a Naïve Bayes classifier with accurate parameters,
 1330 what is the probability of making an error?

- 1331 3. Derive the maximum-likelihood estimate for the parameter μ in Naïve Bayes.
- 1332 4. The classification models in the text have a vector of weights for each possible label.
 1333 While this is notationally convenient, it is overdetermined: for any linear classifier
 1334 that can be obtained with $K \times V$ weights, an equivalent classifier can be constructed
 1335 using $(K - 1) \times V$ weights.
- 1336 a) Describe how to construct this classifier. Specifically, if given a set of weights
 1337 $\boldsymbol{\theta}$ and a feature function $\mathbf{f}(\mathbf{x}, y)$, explain how to construct alternative weights
 1338 and feature function $\boldsymbol{\theta}'$ and $\mathbf{f}'(\mathbf{x}, y)$, such that,

$$\forall y, y' \in \mathcal{Y}, \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y') = \boldsymbol{\theta}' \cdot \mathbf{f}'(\mathbf{x}, y) - \boldsymbol{\theta}' \cdot \mathbf{f}'(\mathbf{x}, y'). \quad [2.82]$$

- 1339 b) Explain how your construction justifies the well-known alternative form for
 1340 binary logistic regression, $\Pr(Y = 1 | \mathbf{x}; \boldsymbol{\theta}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}' \cdot \mathbf{x})} = \sigma(\boldsymbol{\theta}' \cdot \mathbf{x})$, where σ
 1341 is the sigmoid function.
- 1342 5. Suppose you have two labeled datasets D_1 and D_2 , with the same features and la-
 1343 bels.

- 1344 • Let $\theta^{(1)}$ be the unregularized logistic regression (LR) coefficients from training
 1345 on dataset D_1 .
 1346 • Let $\theta^{(2)}$ be the unregularized LR coefficients (same model) from training on
 1347 dataset D_2 .
 1348 • Let θ^* be the unregularized LR coefficients from training on the combined
 1349 dataset $D_1 \cup D_2$.

Under these conditions, prove that for any feature j ,

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

- 1350
- 1351 6. Let $\hat{\theta}$ be the solution to an unregularized logistic regression problem, and let θ^* be
 1352 the solution to the same problem, with L_2 regularization. Prove that $\|\theta^*\|_2^2 \leq \|\hat{\theta}\|_2^2$.
 1353 7. As noted in the discussion of averaged perceptron in § 2.2.2, the computation of the
 1354 running sum $\mathbf{m} \leftarrow \mathbf{m} + \theta$ is unnecessarily expensive, requiring $K \times V$ operations.
 1355 Give an alternative way to compute the averaged weights $\bar{\theta}$, with complexity that is
 1356 independent of V and linear in the sum of feature sizes $\sum_{i=1}^N |\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)})|$.
 1357 8. Consider a dataset that is comprised of two identical instances $\mathbf{x}^{(1)} = \mathbf{x}^{(2)}$ with
 1358 distinct labels $y^{(1)} \neq y^{(2)}$. Assume all features are binary, $x_j \in \{0, 1\}$ for all j .

1359 Now suppose that the averaged perceptron always trains on the instance $(\mathbf{x}^{i(t)}, y^{i(t)})$,
 1360 where $i(t) = 2 - (t \bmod 2)$, which is 1 when the training iteration t is odd, and 2
 1361 when t is even. Further suppose that learning terminates under the following con-
 1362 dition:

$$\epsilon \geq \max_j \left| \frac{1}{t} \sum_t \theta_j^{(t)} - \frac{1}{t-1} \sum_t \theta_j^{(t-1)} \right|. \quad [2.83]$$

1363 In words, the algorithm stops when the largest change in the averaged weights is
 1364 less than or equal to ϵ . Compute the number of iterations before the averaged per-
 1365 ceptron terminates.

- 1366 9. Prove that the margin loss is convex in θ . Use this definition of the margin loss:

$$L(\theta) = -\theta \cdot \mathbf{f}(\mathbf{x}, y^*) + \max_y \theta \cdot \mathbf{f}(\mathbf{x}, y) + c(y^*, y), \quad [2.84]$$

1367 where y^* is the gold label. As a reminder, a function f is convex iff,

$$f(\alpha x_1 + (1 - \alpha)x_2) \leq \alpha f(x_1) + (1 - \alpha)f(x_2), \quad [2.85]$$

1368 for any x_1, x_2 and $\alpha \in [0, 1]$.

- 1369 10. If a function f is m -strongly convex, then for some $m > 0$, the following inequality
1370 holds for all x and y on the domain of the function:

$$f(y) \leq f(x) + (\nabla_x f) \cdot (y - x) + \frac{m}{2} \|y - x\|_2^2. \quad [2.86]$$

1371 Let $f(x) = L(\boldsymbol{\theta}^{(t)})$, the loss of the classifier at iteration t of gradient descent, and let
1372 $f(y) = L(\boldsymbol{\theta}^{(t+1)})$. Assuming the loss function is m -convex, prove that $L(\boldsymbol{\theta}^{(t+1)}) \leq$
1373 $L(\boldsymbol{\theta}^{(t)})$ for an appropriate constant learning rate η , which will depend on m . Explain
1374 why this implies that gradient descent converges when applied to an m -strongly
1375 convex loss function with a unique minimum.

1376 **Chapter 3**

1377 **Nonlinear classification**

1378 Linear classification may seem like all we need for natural language processing. The bag-
1379 of-words representation is inherently high dimensional, and the number of features is
1380 often larger than the number of labeled training instances. This means that it is usually
1381 possible to find a linear classifier that perfectly fits the training data, or even to fit any ar-
1382bitrary labeling of the training instances! Moving to nonlinear classification may therefore
1383 only increase the risk of overfitting. Furthermore, for many tasks, **lexical features** (words)
1384 are meaningful in isolation, and can offer independent evidence about the instance label
1385 — unlike computer vision, where individual pixels are rarely informative, and must be
1386 evaluated holistically to make sense of an image. For these reasons, natural language
1387 processing has historically focused on linear classification.

1388 But in recent years, nonlinear classifiers have swept through natural language pro-
1389 cessing, and are now the default approach for many tasks (Manning, 2016). There are at
1390 least three reasons for this change.

- 1391 • There have been rapid advances in **deep learning**, a family of nonlinear meth-
1392 ods that learn complex functions of the input through multiple layers of computa-
1393 tion (Goodfellow et al., 2016).
- 1394 • Deep learning facilitates the incorporation of **word embeddings**, which are dense
1395 vector representations of words. Word embeddings can be learned from large amounts
1396 of unlabeled data, and enable generalization to words that do not appear in the an-
1397notated training data (word embeddings are discussed in detail in chapter 14).
- 1398 • While CPU speeds have plateaued, there have been rapid advances in specialized
1399 hardware called graphics processing units (GPUs), which have become faster, cheaper,
1400 and easier to program. Many deep learning models can be implemented efficiently
1401 on GPUs, offering substantial performance improvements over CPU-based comput-
1402 ing.

1403 This chapter focuses on **neural networks**, which are the dominant approach for non-
 1404 linear classification in natural language processing today.¹ Historically, a few other non-
 1405 linear learning methods have been applied to language data.

- 1406 • **Kernel methods** are generalizations of the **nearest-neighbor** classification rule, which
 1407 classifies each instance by the label of the most similar example in the training set.
 1408 The application of the **kernel support vector machine** to information extraction is
 1409 described in chapter 17.
- 1410 • **Decision trees** classify instances by checking a set of conditions. Scaling decision
 1411 trees to bag-of-words inputs is difficult, but decision trees have been successful in
 1412 problems such as coreference resolution (chapter 15), where more compact feature
 1413 sets can be constructed (Soon et al., 2001).
- 1414 • **Boosting** and related **ensemble methods** work by combining the predictions of sev-
 1415 eral “weak” classifiers, each of which may consider only a small subset of features.
 1416 Boosting has been successfully applied to text classification (Schapire and Singer,
 1417 2000) and syntactic analysis (Abney et al., 1999), and remains one of the most suc-
 1418 cessful methods on machine learning competition sites such as Kaggle (Chen and
 1419 Guestrin, 2016).

1420 Hastie et al. (2009) provide an excellent overview of these techniques.

1421 3.1 Feedforward neural networks

1422 Consider the problem of building a classifier for movie reviews. The goal is to predict a
 1423 label $y \in \{\text{GOOD}, \text{BAD}, \text{OKAY}\}$ from a representation of the text of each document, x . But
 1424 what makes a good movie? The story, acting, cinematography, editing, soundtrack, and
 1425 so on. Now suppose the training set contains labels for each of these additional features,
 1426 $z = [z_1, z_2, \dots, z_{K_z}]^\top$. With a training set of such information, we could build a two-step
 1427 classifier:

- 1428 1. **Use the text x to predict the features z .** Specifically, train a logistic regression clas-
 1429 sifier to compute $p(z_k | x)$, for each $k \in \{1, 2, \dots, K_z\}$.
- 1430 2. **Use the features z to predict the label y .** Again, train a logistic regression classifier
 1431 to compute $p(y | z)$. On test data, z is unknown, so we will use the probabilities
 1432 $p(z | x)$ from the first layer as the features.

1433 This setup is shown in Figure 3.1, which describes the proposed classifier in a **computation**
 1434 **graph**: the text features x are connected to the middle layer z , which is connected to
 1435 the label y .

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

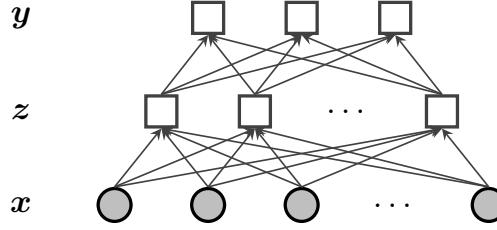


Figure 3.1: A feedforward neural network. Shaded circles indicate observed features, usually words; squares indicate nodes in the computation graph, which are computed from the information carried over the incoming arrows.

1436 If we assume that each z_k is binary, $z_k \in \{0, 1\}$, then the probability $p(z_k | x)$ can be
1437 modeled using binary logistic regression:

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

1438 where σ is the **sigmoid** function (shown in Figure 3.2), and the matrix $\Theta^{(x \rightarrow z)} \in \mathbb{R}^{K_z \times V}$ is
1439 constructed by stacking the weight vectors for each z_k ,

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

1440 We will assume that x contains a term with a constant value of 1, so that a corresponding
1441 offset parameter is included in each $\theta_k^{(x \rightarrow z)}$.

1442 The output layer is computed by the multi-class logistic regression probability,

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

1443 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
1444 constructed by concatenation,

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

1445 The vector of probabilities over each possible value of y is denoted,

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

1446 where element j in the output of the **SoftMax** function is computed as in Equation 3.3.

This set of equations defines a multilayer classifier, which can be summarized as,

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

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

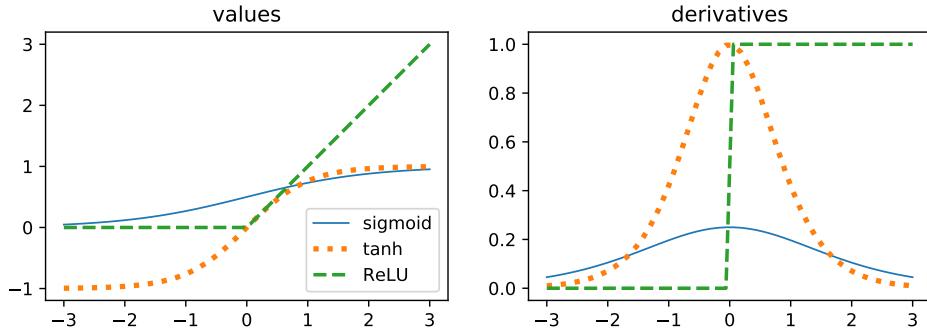


Figure 3.2: The sigmoid, tanh, and ReLU activation functions

1447 where the function σ is now applied **elementwise** to the vector of inner products,

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

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 $\sigma(\theta_k \cdot \mathbf{x})$. The resulting classifier is barely changed:

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

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

1448 This defines a classification model that predicts the label $y \in \mathcal{Y}$ from the base features \mathbf{x} ,
 1449 through a “hidden layer” \mathbf{z} . This is a **feedforward neural network**.²

1450 3.2 Designing neural networks

1451 There several ways to generalize the feedforward neural network.

1452 3.2.1 Activation functions

1453 If the hidden layer is viewed as a set of latent features, then the sigmoid function repre-
 1454 sents the extent to which each of these features is “activated” by a given input. However,
 1455 the hidden layer can be regarded more generally as a nonlinear transformation of the in-
 1456 put. This opens the door to many other activation functions, some of which are shown in
 1457 Figure 3.2. At the moment, the choice of activation functions is more art than science, but
 1458 a few points can be made about the most popular varieties:

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

- The range of the sigmoid function is $(0, 1)$. The bounded range ensures that a cascade of sigmoid functions will not “blow up” to a huge output, and this is important 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, leading to small gradients on its own parameters. Several other activation functions are reviewed in the textbook 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 $z^{(d)}$ acting as the input to the next layer, $z^{(d+1)}$. As the total number of nodes in the network increases, so does its capacity to learn complex functions of the input. Given a fixed number of nodes, one must decide 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)

1488 It is also possible to “short circuit” a hidden layer, by propagating information directly
 1489 from the input to the next higher level of the network. This is the idea behind **residual net-**
 1490 **works**, which propagate information directly from the input to the subsequent layer (He
 1491 et al., 2016),

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

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 $f(\Theta^{(x \rightarrow z)} x)$ and x ,

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

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

1492 where \odot refers to an elementwise vector product, and 1 is a column vector of ones. As
 1493 before, the sigmoid function is applied elementwise to its input; recall that the output of
 1494 this function is restricted to the range $(0, 1)$. Gating is also used in the **long short-term**
 1495 **memory (LSTM)**, which is discussed in chapter 6. Residual and highway connections
 1496 address a problem with deep architectures: repeated application of a nonlinear activation
 1497 function can make it difficult to learn the parameters of the lower levels of the network,
 1498 which are too distant from the supervision signal.

1499 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**,

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

1500 where $\Theta = \{\Theta^{(x \rightarrow z)}, \Theta^{(z \rightarrow y)}, b\}$ is the entire set of parameters.

This loss can be written alternatively as follows:

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

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

for a survey of these theoretical results. However, depending on the function to be approximated, the width 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 imply that it is possible to *learn* the function using gradient-based optimization.

1501 where $e_{y^{(i)}}$ is a **one-hot vector** of zeros with a value of 1 at position $y^{(i)}$. The inner product
 1502 between $e_{y^{(i)}}$ and \tilde{y} is also called the multinomial **cross-entropy**, and this terminology
 1503 is preferred in many neural networks papers and software packages.

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:

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

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

1504 In regression problems, the output is a scalar or vector (see § 4.1.2). For these problems, a
 1505 typical loss function is the squared error $(y - \hat{y})^2$ or squared norm $\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2$.

1506 3.2.4 Inputs and lookup layers

1507 In text classification, the input layer \mathbf{x} can refer to a bag-of-words vector, where x_j is
 1508 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
 1509 represented by the vector $\theta_j^{(x \rightarrow z)}$. This vector is sometimes described as the **embedding** of
 1510 word j , and can be learned from unlabeled data, using techniques discussed in chapter 14.
 1511 The columns of $\Theta^{(x \rightarrow z)}$ are each K_z -dimensional word embeddings.

1512 Chapter 2 presented an alternative view of text documents, as a sequence of word
 1513 tokens, w_1, w_2, \dots, w_M . In a neural network, each word token w_m is represented with a
 1514 one-hot vector, e_{w_m} , with dimension V . The matrix-vector product $\Theta^{(x \rightarrow z)} e_{w_m}$ returns
 1515 the embedding of word w_m . The complete document can be represented by horizontally
 1516 concatenating these one-hot vectors, $\mathbf{W} = [e_{w_1}, e_{w_2}, \dots, e_{w_M}]$, and the bag-of-words rep-
 1517 resentation can be recovered from the matrix-vector product $\mathbf{W}[1, 1, \dots, 1]^\top$, which sums
 1518 each row over the tokens $m = \{1, 2, \dots, M\}$. The matrix product $\Theta^{(x \rightarrow z)} \mathbf{W}$ contains the
 1519 horizontally concatenated embeddings of each word in the document, which will be use-
 1520 ful as the starting point for **convolutional neural networks** (see § 3.4). This is sometimes
 1521 called a **lookup layer**, because the first step is to lookup the embeddings for each word in
 1522 the input text.

1523 3.3 Learning neural networks

The feedforward network in Figure 3.1 can now be written as,

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

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

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

where f is an elementwise activation function, such as σ or ReLU, and $\ell^{(i)}$ is the loss at instance i . The parameters $\Theta^{(x \rightarrow z)}$, $\Theta^{(z \rightarrow y)}$, and b can be estimated using online gradient-based optimization. The simplest such algorithm is stochastic gradient descent, which was discussed in § 2.5. Each parameter is updated by the gradient of the loss,

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

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

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

1524 where $\eta^{(t)}$ is the learning rate on iteration t , $\ell^{(i)}$ is the loss on instance (or minibatch) i ,
1525 and $\boldsymbol{\theta}_n^{(x \rightarrow z)}$ is column n 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 similar to the gradients in logistic regression. For $\boldsymbol{\theta}_k^{(z \rightarrow y)}$, the gradient is,

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

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

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

1526 where $\delta(j = y^{(i)})$ is a function that returns one when $j = y^{(i)}$, and zero otherwise. The
1527 gradient $\nabla_{\mathbf{b}} \ell^{(i)}$ is similar to Equation 3.29.

The gradients on the input layer weights $\Theta^{(x \rightarrow z)}$ are obtained by 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,

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

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

1528 For intuition, consider each of the terms in the product.

- 1529 • 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
1530 doesn't matter how z_k is computed, and so $\frac{\partial \ell^{(i)}}{\partial \theta_{n,k}^{(x \rightarrow z)}} \rightarrow 0$.
- 1531 • If z_k is near 1 or 0, then the curve of the sigmoid function is nearly flat (Figure 3.2),
1532 and changing the inputs will make little local difference. The term $z_k \times (1 - z_k)$ is
1533 maximized at $z_k = \frac{1}{2}$, where the slope of the sigmoid function is steepest.
- 1534 • If $x_n = 0$, then it does not matter how we set the weights $\theta_{n,k}^{(x \rightarrow z)}$, so $\frac{\partial \ell^{(i)}}{\partial \theta_{n,k}^{(x \rightarrow z)}} = 0$.

1535 3.3.1 Backpropagation

1536 The equations above rely on the chain rule to compute derivatives of the loss with respect
1537 to each parameter of the model. Furthermore, local derivatives are frequently reused: for
1538 example, $\frac{\partial \ell^{(i)}}{\partial z_k}$ is reused in computing the derivatives with respect to each $\theta_{n,k}^{(x \rightarrow z)}$. These
1539 terms should therefore be computed once, and then cached. Furthermore, we should only
1540 compute any derivative once we have already computed all of the necessary “inputs”
1541 demanded by the chain rule of differentiation. This combination of sequencing, caching,
1542 and differentiation is known as **backpropagation**. It can be generalized to any directed
1543 acyclic **computation graph**.

1544 A computation graph is a declarative representation of a computational process. At
1545 each node t , compute a value v_t by applying a function f_t to a (possibly empty) list of
1546 parent nodes, π_t . For example, in a feedforward network with one hidden layer, there are
1547 nodes for the input $\mathbf{x}^{(i)}$, the hidden layer \mathbf{z} , the predicted output $\tilde{\mathbf{y}}$, and the parameters
1548 $\{\Theta^{(x \rightarrow z)}, \Theta^{(z \rightarrow y)}, \mathbf{b}\}$. During training, there is also a node for the label $\mathbf{y}^{(i)}$ and the loss
1549 $\ell^{(i)}$. The predicted output $\tilde{\mathbf{y}}$ is one of the parents of the loss (the other is the label $\mathbf{y}^{(i)}$); its
1550 parents include $\Theta^{(z \rightarrow y)}$ and \mathbf{z} , and so on.

1551 Computation graphs include three types of nodes:

1552 **Variables.** The variables include the inputs \mathbf{x} , the hidden nodes \mathbf{z} , the outputs \mathbf{y} , and
1553 the loss function. Inputs are variables that do not have parents. Backpropagation

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 $x^{(i)}$.

computes the gradients with respect to all variables except the inputs, but does not update the variables during learning.

1556 **Parameters.** In a feedforward network, the parameters include the weights and offsets.
1557 Parameter nodes do not have parents, and they are updated during learning.

1558 Objective. The *objective* node is not the parent of any other node. Backpropagation begins
1559 by computing the gradient with respect to this node.

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

While the gradients with respect to each parameter may be complex, they are composed of products of simple parts. For many networks, all gradients can be computed through **automatic differentiation**. This means that end users need only specify the feed-forward computation, and the gradients necessary for learning can be obtained automatically. There are many software libraries that perform automatic differentiation on computation graphs, such as Torch (Collobert et al., 2011), TensorFlow (Abadi et al., 2016), and DyNet (Neubig et al., 2017). One important distinction between these libraries is whether they support **dynamic computation graphs**, in which the structure of the computation graph varies across instances. Static computation graphs are compiled in advance, and can be applied to fixed-dimensional data, such as bag-of-words vectors. In many natu-

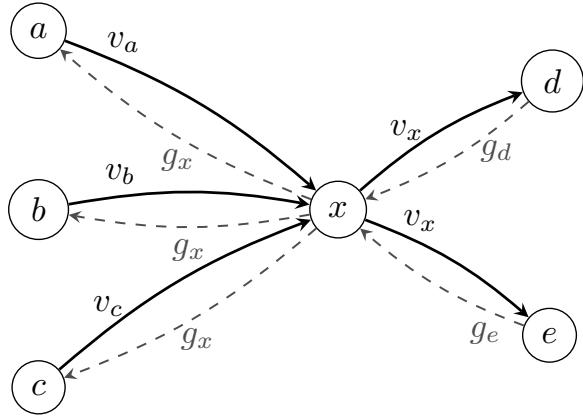


Figure 3.3: Backpropagation at a single node x in the computation graph. The values of the predecessors v_a, v_b, v_c are the inputs to x , which computes v_x , and passes it on to the successors d and e . The gradients at the successors g_d and g_e are passed back to x , where they are incorporated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

1577 ral language processing problems, each input has a distinct structure, requiring a unique
 1578 computation graph.

1579 3.3.2 Regularization and dropout

1580 In linear classification, overfitting was addressed by augmenting the objective with a reg-
 1581 ularization term, $\lambda \|\boldsymbol{\theta}\|_2^2$. This same approach can be applied to feedforward neural net-
 1582 works, penalizing each matrix of weights:

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

1583 where $\|\boldsymbol{\Theta}\|_F^2 = \sum_{i,j} \theta_{i,j}^2$ is the squared **Frobenius norm**, which generalizes the L_2 norm
 1584 to matrices. The bias parameters b are not regularized, as they do not contribute to the
 1585 sensitivity of the classifier to the inputs. In gradient-based optimization, the practical
 1586 effect of Frobenius norm regularization is that the weights “decay” towards zero at each
 1587 update, motivating the alternative name **weight decay**.

1588 Another approach to controlling model complexity is **dropout**, which involves ran-
 1589 domly setting some computation nodes to zero during training (Srivastava et al., 2014).
 1590 For example, in the feedforward network, on each training instance, with probability ρ we
 1591 set each input x_n and each hidden layer node z_k to zero. Srivastava et al. (2014) recom-
 1592 mend $\rho = 0.5$ for hidden units, and $\rho = 0.2$ for input units. Dropout is also incorporated

in the gradient computation, so if node z_k is dropped, then none of the weights $\theta_k^{(x \rightarrow z)}$ will be updated for this instance. Dropout prevents the network from learning to depend too 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 $\tilde{z} = [\tilde{z}_{\pi(1)}, \tilde{z}_{\pi(2)}, \dots, \tilde{z}_{\pi(K_z)}]$. 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 $x = (0, 0)$. 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 learn-

1631 ing will slow to a crawl as the gradient goes to zero. However, the noise introduced by
 1632 stochastic gradient descent, and by feature noising techniques such as dropout, can help
 1633 online optimization to escape saddle points and find high-quality optima (Ge et al., 2015).
 1634 Other techniques address saddle points directly, using local reconstructions of the Hessian
 1635 matrix (Dauphin et al., 2014) or higher-order derivatives (Anandkumar and Ge, 2016).

1636 **3.3.4 Tricks**

1637 Getting neural networks to work effectively sometimes requires heuristic “tricks” (Bottou,
 1638 2012; Goodfellow et al., 2016; Goldberg, 2017b). This section presents some tricks that are
 1639 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]

1640 For the weights leading to a ReLU activation function, He et al. (2015) use similar argu-
 1641 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:

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

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

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

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

1642 The matrix \mathbf{U} contains the **singular vectors** of \mathbf{A} , and is guaranteed to be orthonormal.
 1643 For more on singular value decomposition, see chapter 14.

1644 Even with careful initialization, there can still be significant variance in the final re-
 1645 sults. It can be useful to make multiple training runs, and select the one with the best
 1646 performance on a heldout development set.

1647 **Clipping and normalizing the gradients** As already discussed, the magnitude of the
 1648 gradient can pose problems for learning: too large, and learning can diverge, with suc-
 1649 cessive updates thrashing between increasingly extreme values; too small, and learning can
 1650 grind to a halt. Several heuristics have been proposed to address this issue.

1651 • In **gradient clipping** (Pascanu et al., 2013), an upper limit is placed on the norm of
 1652 the gradient, and the gradient is rescaled when this limit is exceeded,

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

1653 • In **batch normalization** (Ioffe and Szegedy, 2015), the inputs to each computation
 1654 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:

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

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

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

1653 Empirically, this speeds convergence of deep architectures. One explanation is that
 1654 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:

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

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

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

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

1655 Layer normalization has similar motivations to batch normalization, but it can be
 1656 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,

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

1657 where $g_{t,j}$ is the gradient with respect to parameter j at time t , and $\beta \in [0, 1]$. This term
 1658 places more emphasis on recent gradients, and is employed in the **AdaDelta** (Zeiler, 2012)
 1659 and **Adam** (Kingma and Ba, 2014) optimizers. Online optimization and its theoretical
 1660 background are reviewed by Bottou et al. (2016). **Early stopping**, mentioned in § 2.2.2,
 1661 can help to avoid overfitting, by terminating training after reaching a plateau in the per-
 1662 formance on a heldout validation set.

1663 3.4 Convolutional neural networks

1664 A basic weakness of the bag-of-words model is its inability to account for the ways in
 1665 which words combine to create meaning, including even simple reversals such as *not*
 1666 *pleasant*, *hardly a generous offer*, and *I wouldn't mind missing the flight*. Computer vision
 1667 faces the related challenge of identifying the semantics of images from pixel features
 1668 that are uninformative in isolation. An earlier generation of computer vision research
 1669 focused on designing *filters* to aggregate local pixel-level features into more meaningful
 1670 representations, such as edges and corners (e.g., Canny, 1987). Similarly, earlier NLP re-
 1671 search attempted to capture multiword linguistic phenomena by hand-designed lexical
 1672 patterns (Hobbs et al., 1997). In both cases, the output of the filters and patterns could

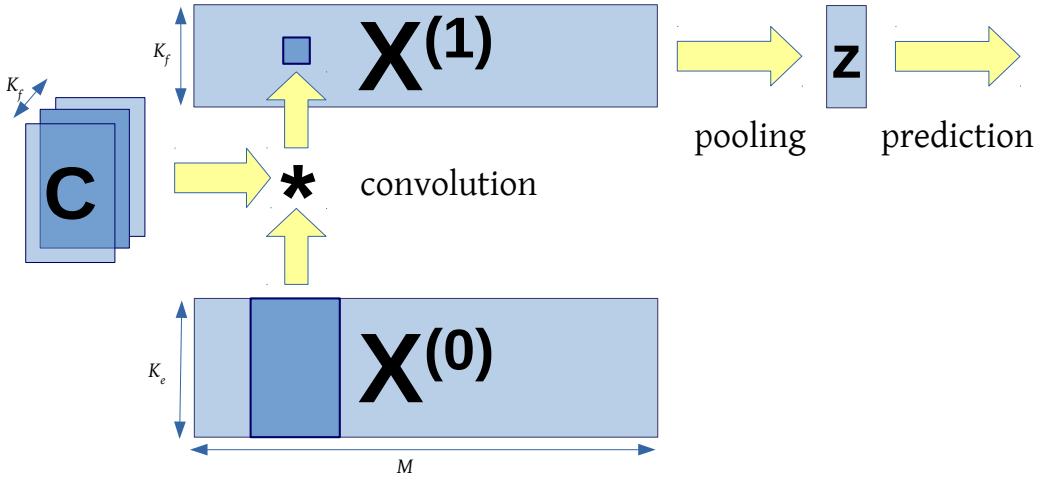


Figure 3.4: A convolutional neural network for text classification

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 , we 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,

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

1681 thereby representing trigram units like *not so unpleasant*. In **one-dimensional convolution**,
 1682 each filter matrix $\mathbf{C}^{(k)}$ is constrained to have non-zero values only at row k (Kalchbrenner
 1683 et al., 2014). This means that each dimension of the word embedding is processed
 1684 by a separate filter, and it implies that $K_f = K_e$.

1685 To deal with the beginning and end of the input, the base matrix $\mathbf{X}^{(0)}$ may be padded
 1686 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
 1687 than the input; this is known as **narrow convolution**. The filter matrices need not have
 1688 identical filter widths, so more generally we could write h_k to indicate width of filter
 1689 $\mathbf{C}^{(k)}$. As suggested by the notation $\mathbf{X}^{(0)}$, multiple layers of convolution may be applied,
 1690 so that $\mathbf{X}^{(d)}$ is the input to $\mathbf{X}^{(d+1)}$.

1691 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
 1692 positions to obtain a fixed-length representation. This can be done by a **pooling** operation,
 1693 such as max-pooling (Collobert et al., 2011) or average-pooling,

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

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

1692 The vector \mathbf{z} can now act as a layer in a feedforward network, culminating in a prediction
 1693 \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 :

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

1694 The computer vision literature has produced a huge variety of convolutional architectures,
 1695 and many of these innovations can be applied to text data. One avenue for
 1696 improvement is more complex pooling operations, such as k -max pooling (Kalchbrenner
 1697 et al., 2014), which returns a matrix of the k largest values for each filter. Another innovation
 1698 is the use of **dilated convolution** to build multiscale representations (Yu and Koltun,
 1699 2016). At each layer, the convolutional operator applied in *strides*, skipping ahead by s
 1700 steps after each feature. As we move up the hierarchy, each layer is s times smaller than
 1701 the layer below it, effectively summarizing the input (Kalchbrenner et al., 2016; Strubell
 1702 et al., 2017). This idea is shown in Figure 3.5. Multi-layer convolutional networks can also

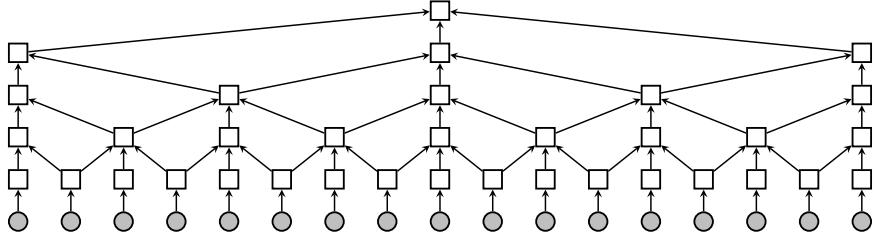


Figure 3.5: A dilated convolutional neural network captures progressively larger context through recursive application of the convolutional operator

1703 be augmented with “shortcut” connections, as in the residual network from § 3.2.2 (Johnson and Zhang, 2017).
 1704

1705 Additional resources

1706 The deep learning textbook by Goodfellow et al. (2016) covers many of the topics in this
 1707 chapter in more detail. For a comprehensive review of neural networks in natural lan-
 1708 guage processing, see Goldberg (2017b). A seminal work on deep learning in natural
 1709 language processing is the aggressively titled “Natural Language Processing (Almost)
 1710 from Scratch”, which uses convolutional neural networks to perform a range of language
 1711 processing tasks (Collobert et al., 2011), although there is earlier work (e.g., Henderson,
 1712 2004). This chapter focuses on feedforward and convolutional neural networks, but recur-
 1713 rent neural networks are one of the most important deep learning architectures for natural
 1714 language processing. They are covered extensively in chapters 6 and 7.

1715 The role of deep learning in natural language processing research has caused angst
 1716 in some parts of the natural language processing research community (e.g., Goldberg,
 1717 2017a), especially as some of the more zealous deep learning advocates have argued that
 1718 end-to-end learning from “raw” text can eliminate the need for linguistic constructs such
 1719 as sentences, phrases, and even words (Zhang et al., 2015, originally titled *Text understand-
 1720 ing from scratch*). These developments were surveyed by Manning (2016). While reports of
 1721 the demise of linguistics in natural language processing remain controversial at best, deep
 1722 learning and backpropagation have become ubiquitous in both research and applications.

1723 Exercises

- 1724 1. a) Draw the computation graph for a feedforward network with a single hidden
 1725 layer. You may represent the vector of values at each layer as a single node.
 1726 Don’t forget to include the parameters, the label, and the loss.
 1727 b) Update your computation graph to include a residual connection.

- 1728 c) Update your computation graph to include a highway connection.
- 1729 2. Prove that the softmax and sigmoid functions are equivalent when the number of
 1730 possible labels is two. Specifically, for any $\Theta^{(z \rightarrow y)}$ (omitting the offset b for simplic-
 1731 ity), show how to construct a vector of weights θ such that,

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

- 1732 3. Convolutional neural networks often aggregate across words by using **max pooling**
 1733 (Equation 3.55 in § 3.4). A potential concern is that there is zero gradient with respect
 1734 to the parts of the input that are involved in the maximum. The following questions
 1735 consider the gradient with respect to an element of the input, $x_{m,k}^{(0)}$, and they assume
 1736 that all parameters are independently distributed.

- 1737 a) First consider a minimal network, with $z = \text{MaxPool}(\mathbf{X}^{(0)})$. What is the prob-
 1738 ability that the gradient $\frac{\partial \ell}{\partial x_{m,k}^{(0)}}$ is non-zero?
- 1739 b) Now consider a two-level network, with $\mathbf{X}^{(1)} = f(\mathbf{b} + \mathbf{C} * \mathbf{X}^{(0)})$. Express the
 1740 probability that the gradient $\frac{\partial \ell}{\partial x_{m,k}^{(0)}}$ is non-zero, in terms of the input length M ,
 1741 the filter size n , and the number of filters K_f .
- 1742 c) Using a calculator, work out the probability for the case $M = 128, n = 4, K_f =$
 1743 32.
- 1744 d) Now consider a three-level network, $\mathbf{X}^{(2)} = f(\mathbf{b} + \mathbf{C} * \mathbf{X}^{(1)})$. Give the general
 1745 equation for the probability that $\frac{\partial \ell}{\partial x_{m,k}^{(0)}}$ is non-zero, and compute the numerical
 1746 probability for the scenario in the previous part, assuming $K_f = 32$ and $n = 4$
 1747 at both levels.

- 1748 4. Design a feedforward network to compute the XOR function:

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

1749 Your network should have a single output node which uses the Sign activation func-
 1750 tion, $f(x) = \begin{cases} 1, & x > 0 \\ -1, & x \leq 0 \end{cases}$. Use a single hidden layer, with ReLU activation func-
 1751 tions. Describe all weights and offsets.

- 1752 5. Consider the same network as above (with ReLU activations for the hidden layer),
 1753 with an arbitrary differentiable loss function $\ell(y^{(i)}, \tilde{y})$, where \tilde{y} is the activation of
 1754 the output node. Suppose all weights and offsets are initialized to zero. Prove that
 1755 gradient-based optimization cannot learn the desired function from this initializa-
 1756 tion.
- 1757 6. The simplest solution to the previous problem relies on the use of the ReLU activa-
 1758 tion function at the hidden layer. Now consider a network with arbitrary activations
 1759 on the hidden layer. Show that if the initial weights are any uniform constant, then
 1760 it is not possible to learn the desired function by gradient descent.
- 1761 7. Consider a network in which: the base features are all binary, $\mathbf{x} \in \{0, 1\}^M$; the
 1762 hidden layer activation function is sigmoid, $z_k = \sigma(\theta_k \cdot \mathbf{x})$; and the initial weights
 1763 are sampled independently from a standard normal distribution, $\theta_{j,k} \sim N(0, 1)$.
- 1764 • Show how the probability of a small initial gradient on any weight, $\frac{\partial z_k}{\partial \theta_{j,k}} < \alpha$,
 1765 depends on the size of the input M . **Hint:** use the lower bound,

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

1766 and relate this probability to the variance $V[\theta_k \cdot \mathbf{x}]$.

- 1767 • Design an alternative initialization that removes this dependence.

- 1768 8. The ReLU activation function can lead to “dead neurons”, which can never be acti-
 1769 vated on any input. Consider the following two-layer feedforward network with a
 1770 scalar output y :

$$z_i = \text{ReLU}(\theta_i^{(x \rightarrow z)} \cdot \mathbf{x} + b_i) \quad [3.61]$$

$$y = \theta^{(z \rightarrow y)} \cdot z. \quad [3.62]$$

1768 Suppose that the input is a binary vector of observations, $\mathbf{x} \in \{0, 1\}^D$.

- 1769 a) Under what condition is node z_i “dead”? Your answer should be expressed in
 1770 terms of the parameters $\theta_i^{(x \rightarrow z)}$ and b_i .
- 1771 b) Suppose that the gradient of the loss on a given instance is $\frac{\partial \ell}{\partial y} = 1$. Derive the
 1772 gradients $\frac{\partial \ell}{\partial b_i}$ and $\frac{\partial \ell}{\partial \theta_j^{(x \rightarrow z)}}$ for such an instance.
- 1773 c) Using your answers to the previous two parts, explain why a dead neuron can
 1774 never be brought back to life during gradient-based learning.

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

$$\begin{aligned} & \left(\|\tilde{\Theta}^{(x \rightarrow z)} - \Theta^{(x \rightarrow z)}\|_F^2 + \|\tilde{\Theta}^{(z \rightarrow y)} - \Theta^{(z \rightarrow y)}\|_F^2 + \|\tilde{\mathbf{b}} - \mathbf{b}\|_2^2 < \epsilon \right) \\ & \Rightarrow \left(L(\tilde{\Theta}) > L(\Theta) \right) \end{aligned} \quad [3.63]$$

1775 Define the function π as a permutation on the hidden units, as described in § 3.3.3,
 1776 so that for any Θ , $L(\Theta) = L(\Theta_\pi)$. Prove that if a feedforward network has a local
 1777 optimum in the sense of Equation 3.63, then its loss is not a convex function of the
 1778 parameters Θ , using the definition of convexity from § 2.3

- 1779 10. Consider a network with a single hidden layer, and a single output,

$$y = \boldsymbol{\theta}^{(z \rightarrow y)} \cdot g(\boldsymbol{\Theta}^{(x \rightarrow z)} \mathbf{x}). \quad [3.64]$$

1780 Assume that g is the ReLU function. Show that for any matrix of weights $\boldsymbol{\Theta}^{(x \rightarrow z)}$, it
 1781 is permissible to rescale each row to have a norm of one, because an identical output
 1782 can be obtained by finding a corresponding rescaling of $\boldsymbol{\theta}^{(z \rightarrow y)}$.

¹⁷⁸³ Chapter 4

¹⁷⁸⁴ Linguistic applications of ¹⁷⁸⁵ classification

¹⁷⁸⁶ Having learned some techniques for classification, this chapter shifts the focus from mathematics
¹⁷⁸⁷ to linguistic applications. Later in the chapter, we will consider the design decisions involved in text classification, as well as evaluation practices.

¹⁷⁸⁹ 4.1 Sentiment and opinion analysis

¹⁷⁹⁰ A popular application of text classification is to automatically determine the **sentiment**
¹⁷⁹¹ or **opinion polarity** of documents such as product reviews and social media posts. For
¹⁷⁹² example, marketers are interested to know how people respond to advertisements, ser-
¹⁷⁹³ vices, and products (Hu and Liu, 2004); social scientists are interested in how emotions
¹⁷⁹⁴ are affected by phenomena such as the weather (Hannak et al., 2012), and how both opin-
¹⁷⁹⁵ ions and emotions spread over social networks (Coviello et al., 2014; Miller et al., 2011).
¹⁷⁹⁶ In the field of **digital humanities**, literary scholars track plot structures through the flow
¹⁷⁹⁷ of sentiment across a novel (Jockers, 2015).¹

¹⁷⁹⁸ Sentiment analysis can be framed as a direct application of document classification,
¹⁷⁹⁹ assuming reliable labels can be obtained. In the simplest case, sentiment analysis is a
¹⁸⁰⁰ two or three-class problem, with sentiments of POSITIVE, NEGATIVE, and possibly NEU-
¹⁸⁰¹ TRAL. Such annotations could be annotated by hand, or obtained automatically through
¹⁸⁰² a variety of means:

- ¹⁸⁰³ • Tweets containing happy emoticons can be marked as positive, sad emoticons as
¹⁸⁰⁴ 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).

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

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

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

- 1819 (4.1) That's not bad for the first day.
- 1820 (4.2) This is not the worst thing that can happen.
- 1821 (4.3) It would be nice if you acted like you understood.
- 1822 (4.4) There is no reason at all to believe that the polluters are suddenly going to be-
1823 come reasonable. (Wilson et al., 2005)
- 1824 (4.5) This film should be brilliant. The actors are first grade. Stallone plays a happy,
1825 wonderful man. His sweet wife is beautiful and adores him. He has a fascinat-
1826 ing gift for living life fully. It sounds like a great plot, **however**, the film is a
1827 failure. (Pang et al., 2002)

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

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

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

4.1.1 Related problems

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

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

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

- 1858 (4.6) The vodka was good, but the meat was rotten.
- 1859 (4.7) Go to Heaven for the climate, Hell for the company. —Mark Twain

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

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

1872 **Emotion classification** While sentiment analysis is framed in terms of positive and neg-
 1873 ative categories, psychologists generally regard **emotion** as more multifaceted. For ex-
 1874 ample, Ekman (1992) argues that there are six basic emotions — happiness, surprise, fear,
 1875 sadness, anger, and contempt — and that they are universal across human cultures. Alm
 1876 et al. (2005) build a linear classifier for recognizing the emotions expressed in children’s
 1877 stories. The ultimate goal of this work was to improve text-to-speech synthesis, so that
 1878 stories could be read with intonation that reflected the emotional content. They used bag-
 1879 of-words features, as well as features capturing the story type (e.g., jokes, folktales), and
 1880 structural features that reflect the position of each sentence in the story. The task is diffi-
 1881 cult: even human annotators frequently disagreed with each other, and the best classifiers
 1882 achieved accuracy between 60-70%.

1883 4.1.2 Alternative approaches to sentiment analysis

1884 **Regression** A more challenging version of sentiment analysis is to determine not just
 1885 the class of a document, but its rating on a numerical scale (Pang and Lee, 2005). If the
 1886 scale is continuous, it is most natural to apply **regression**, identifying a set of weights θ
 1887 that minimize the squared error of a predictor $\hat{y} = \theta \cdot x + b$, where b is an offset. This
 1888 approach is called **linear regression**, and sometimes **least squares**, because the regression
 1889 coefficients θ are determined by minimizing the squared error, $(y - \hat{y})^2$. If the weights are
 1890 regularized using a penalty $\lambda \|\theta\|_2^2$, then it is **ridge regression**. Unlike logistic regression,
 1891 both linear regression and ridge regression can be solved in closed form as a system of
 1892 linear equations.

1893 **Ordinal ranking** In many problems, the labels are ordered but discrete: for example,
 1894 product reviews are often integers on a scale of 1 – 5, and grades are on a scale of A – F.
 1895 Such problems can be solved by discretizing the score $\theta \cdot x$ into “ranks”,

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

1896 where $\mathbf{b} = [b_1 = -\infty, b_2, b_3, \dots, b_K]$ is a vector of boundaries. It is possible to learn the
 1897 weights and boundaries simultaneously, using a perceptron-like algorithm (Crammer and
 1898 Singer, 2001).

1899 **Lexicon-based classification** Sentiment analysis is one of the only NLP tasks where
 1900 hand-crafted feature weights are still widely employed. In **lexicon-based classification** (Taboada
 1901 et al., 2011), the user creates a list of words for each label, and then classifies each docu-
 1902 ment based on how many of the words from each list are present. In our linear classifica-
 1903 tion framework, this is equivalent to choosing the following weights:

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

1904 where \mathcal{L}_y is the lexicon for label y . Compared to the machine learning classifiers discussed
 1905 in the previous chapters, lexicon-based classification may seem primitive. However, su-
 1906 pervised machine learning relies on large annotated datasets, which are time-consuming
 1907 and expensive to produce. If the goal is to distinguish two or more categories in a new
 1908 domain, it may be simpler to start by writing down a list of words for each category.

1909 An early lexicon was the *General Inquirer* (Stone, 1966). Today, popular sentiment lex-
 1910 cons include sentiwordnet (Esuli and Sebastiani, 2006) and an evolving set of lexicons
 1911 from Liu (2015). For emotions and more fine-grained analysis, *Linguistic Inquiry and Word*
 1912 *Count* (LIWC) provides a set of lexicons (Tausczik and Pennebaker, 2010). The MPQA lex-
 1913 icon indicates the polarity (positive or negative) of 8221 terms, as well as whether they are
 1914 strongly or weakly subjective (Wiebe et al., 2005). A comprehensive comparison of senti-
 1915 ment lexicons is offered by Ribeiro et al. (2016). Given an initial **seed lexicon**, it is possible
 1916 to automatically expand the lexicon by looking for words that frequently co-occur with
 1917 words in the seed set (Hatzivassiloglou and McKeown, 1997; Qiu et al., 2011).

1918 4.2 Word sense disambiguation

1919 Consider the the following headlines:

- 1920 (4.8) Iraqi head seeks arms
- 1921 (4.9) Prostitutes appeal to Pope
- 1922 (4.10) Drunk gets nine years in violin case²

1923 These headlines are ambiguous because they contain words that have multiple mean-
 1924 ings, or **senses**. Word sense disambiguation is the problem of identifying the intended
 1925 sense of each word token in a document. Word sense disambiguation is part of a larger
 1926 field of research called **lexical semantics**, which is concerned with meanings of the words.

1927 At a basic level, the problem of word sense disambiguation is to identify the correct
 1928 sense for each word token in a document. Part-of-speech ambiguity (e.g., noun versus
 1929 verb) is usually considered to be a different problem, to be solved at an earlier stage.
 1930 From a linguistic perspective, senses are not properties of words, but of **lemmas**, which
 1931 are canonical forms that stand in for a set of inflected words. For example, *arm*/N is a
 1932 lemma that includes the inflected form *arms*/N — the /N indicates that it we are refer-
 1933 ring to the noun, and not its **homonym** *arm*/V, which is another lemma that includes
 1934 the inflected verbs (*arm*/V, *arms*/V, *armed*/V, *arming*/V). Therefore, word sense disam-
 1935 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>

1936 and then choosing the correct sense from the inventory associated with the corresponding
 1937 lemma.³ (Part-of-speech tagging is discussed in § 8.1.)

1938 **4.2.1 How many word senses?**

1939 Words sometimes have many more than two senses, as exemplified by the word *serve*:

- 1940 • [FUNCTION]: *The tree stump served as a table*
- 1941 • [CONTRIBUTE TO]: *His evasive replies only served to heighten suspicion*
- 1942 • [PROVIDE]: *We serve only the rawest fish*
- 1943 • [ENLIST]: *She served in an elite combat unit*
- 1944 • [JAIL]: *He served six years for a crime he didn't commit*
- 1945 • [LEGAL]: *They were served with subpoenas*⁴

1946 These sense distinctions are annotated in **WordNet** (<http://wordnet.princeton.edu>), a lexical semantic database for English. WordNet consists of roughly 100,000 **synsets**,
 1947 which are groups of lemmas (or phrases) that are synonymous. An example synset is
 1948 {*chump*¹, *fool*², *sucker*¹, *mark*⁹}, where the superscripts index the sense of each lemma that
 1949 is included in the synset: for example, there are at least eight other senses of *mark* that
 1950 have different meanings, and are not part of this synset. A lemma is **polysemous** if it
 1951 participates in multiple synsets.

1953 WordNet defines the scope of the word sense disambiguation problem, and, more
 1954 generally, formalizes lexical semantic knowledge of English. (WordNets have been cre-
 1955 ated for a few dozen other languages, at varying levels of detail.) Some have argued
 1956 that WordNet's sense granularity is too fine (Ide and Wilks, 2006); more fundamentally,
 1957 the premise that word senses can be differentiated in a task-neutral way has been criti-
 1958 cized as linguistically naïve (Kilgarriff, 1997). One way of testing this question is to ask
 1959 whether people tend to agree on the appropriate sense for example sentences: accord-
 1960 ing to Mihalcea et al. (2004), people agree on roughly 70% of examples using WordNet
 1961 senses; far better than chance, but less than agreement on other tasks, such as sentiment
 1962 annotation (Wilson et al., 2005).

1963 ***Other lexical semantic relations** Besides **synonymy**, WordNet also describes many
 1964 other lexical semantic relationships, including:

- 1965 • **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.

1988 Using such features, a classifier can be trained from labeled data. A **semantic concor-**
 1989 **dance** is a corpus in which each open-class word (nouns, verbs, adjectives, and adverbs)
 1990 is tagged with its word sense from the target dictionary or thesaurus. SemCor is a seman-
 1991 tic concordance built from 234K tokens of the Brown corpus (Francis and Kucera, 1982),
 1992 annotated as part of the WordNet project (Fellbaum, 2010). SemCor annotations look like
 1993 this:

1994 (4.13) As of Sunday¹_N night¹_N there was⁴_V no word²_N ...,

1995 with the superscripts indicating the annotated sense of each polysemous word, and the
 1996 subscripts indicating the part-of-speech.

1997 As always, supervised classification is only possible if enough labeled examples can
 1998 be accumulated. This is difficult in word sense disambiguation, because each polysemous
 1999 lemma requires its own training set: having a good classifier for the senses of *serve* is no
 2000 help towards disambiguating *plant*. For this reason, unsupervised and **semi-supervised**
 2001 methods are particularly important for word sense disambiguation (e.g., Yarowsky, 1995).
 2002 These methods will be discussed in chapter 5. Unsupervised methods typically lean on
 2003 the heuristic of “one sense per discourse”, which means that a lemma will usually have
 2004 a single, consistent sense throughout any given document (Gale et al., 1992). Based on
 2005 this heuristic, we can propagate information from high-confidence instances to lower-
 2006 confidence instances in the same document (Yarowsky, 1995). Semi-supervised methods
 2007 combine labeled and unlabeled data, and are discussed in more detail in chapter 5.

2008 4.3 Design decisions for text classification

2009 Text classification involves a number of design decisions. In some cases, the design deci-
 2010 sion is clear from the mathematics: if you are using regularization, then a regularization
 2011 weight λ must be chosen. Other decisions are more subtle, arising only in the low level
 2012 “plumbing” code that ingests and processes the raw data. Such decision can be surpris-
 2013 ingly consequential for classification accuracy.

2014 4.3.1 What is a word?

2015 The bag-of-words representation presupposes that extracting a vector of word counts
 2016 from text is unambiguous. But text documents are generally represented as sequences of
 2017 characters (in an encoding such as ascii or unicode), and the conversion to bag-of-words
 2018 presupposes a definition of the “words” that are to be counted.

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? ;)*

2019 4.3.1.1 Tokenization

2020 The first subtask for constructing a bag-of-words vector is **tokenization**: converting the
 2021 text from a sequence of characters to a sequence of **word tokens**. A simple approach is
 2022 to define a subset of characters as whitespace, and then split the text on these tokens.
 2023 However, whitespace-based tokenization is not ideal: we may want to split conjunctions
 2024 like *isn't* and hyphenated phrases like *prize-winning* and *half-asleep*, and we likely want
 2025 to separate words from commas and periods that immediately follow them. At the same
 2026 time, it would be better not to split abbreviations like *U.S.* and *Ph.D.* In languages with
 2027 Roman scripts, tokenization is typically performed using regular expressions, with mod-
 2028 ules designed to handle each of these cases. For example, the `nltk` package includes a
 2029 number of tokenizers (Loper and Bird, 2002); the outputs of four of the better-known tok-
 2030 enizers are shown in Figure 4.1. Social media researchers have found that emoticons and
 2031 other forms of orthographic variation pose new challenges for tokenization, leading to the
 2032 development of special purpose tokenizers to handle these phenomena (O'Connor et al.,
 2033 2010).

2034 Tokenization is a language-specific problem, and each language poses unique chal-
 2035 lenges. For example, Chinese does not include spaces between words, nor any other
 2036 consistent orthographic markers of word boundaries. A “greedy” approach is to scan the
 2037 input for character substrings that are in a predefined lexicon. However, Xue et al. (2003)
 2038 notes that this can be ambiguous, since many character sequences could be segmented in
 2039 multiple ways. Instead, he trains a classifier to determine whether each Chinese character,
 2040 or **hanzi**, is a word boundary. More advanced sequence labeling methods for word seg-
 2041 mentation are discussed in § 8.4. Similar problems can occur in languages with alphabetic
 2042 scripts, such as German, which does not include whitespace in compound nouns, yield-
 2043 ing examples such as *Freundschaftsbezeugungen* (demonstration of friendship) and *Dilett-
 2044 tantenaufdringlichkeiten* (the importunities of dilettantes). As Twain (1997) argues, “*These
 2045 things are not words, they are alphabetic processions.*” Social media raises similar problems
 2046 for English and other languages, with hashtags such as `#TrueLoveInFourWords` requiring
 2047 decomposition for analysis (Brun and Roux, 2014).

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

2048 4.3.1.2 Normalization

2049 After splitting the text into tokens, the next question is which tokens are really distinct.
 2050 Is it necessary to distinguish *great*, *Great*, and *GREAT*? Sentence-initial capitalization may
 2051 be irrelevant to the classification task. Going further, the complete elimination of case
 2052 distinctions will result in a smaller vocabulary, and thus smaller feature vectors. However,
 2053 case distinctions might be relevant in some situations: for example, *apple* is a delicious
 2054 pie filling, while *Apple* is a company that specializes in proprietary dongles and power
 2055 adapters.

2056 For Roman script, case conversion can be performed using unicode string libraries.
 2057 Many scripts do not have case distinctions (e.g., the Devanagari script used for South
 2058 Asian languages, the Thai alphabet, and Japanese kana), and case conversion for all scripts
 2059 may not be available in every programming environment. (Unicode support is an im-
 2060 portant distinction between Python’s versions 2 and 3, and is a good reason for mi-
 2061 grating to Python 3 if you have not already done so. Compare the output of the code
 2062 "\à l\'hôtel".upper() in the two language versions.)⁶

2063 Case conversion is a type of **normalization**, which refers to string transformations that
 2064 remove distinctions that are irrelevant to downstream applications (Sproat et al., 2001).
 2065 Other normalizations include the standardization of numbers (e.g., 1,000 to 1000) and
 2066 dates (e.g., August 11, 2015 to 2015/11/08). Depending on the application, it may even be
 2067 worthwhile to convert all numbers and dates to special tokens, !NUM and !DATE. In social
 2068 media, there are additional orthographic phenomena that may be normalized, such as ex-
 2069 pressive lengthening, e.g., *coooooool* (Aw et al., 2006; Yang and Eisenstein, 2013). Similarly,
 2070 historical texts feature spelling variations that may need to be normalized to a contempo-
 2071 rary standard form (Baron and Rayson, 2008).

2072 A more extreme form of normalization is to eliminate **inflectional affixes**, such as the
 2073 -ed and -s suffixes in English. On this view, *bike*, *bikes*, *biking*, and *biked* all refer to the
 2074 same underlying concept, so they should be grouped into a single feature. A **stemmer** is
 2075 a program for eliminating affixes, usually by applying a series of regular expression sub-
 2076 stitutions. Character-based stemming algorithms are necessarily approximate, as shown

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

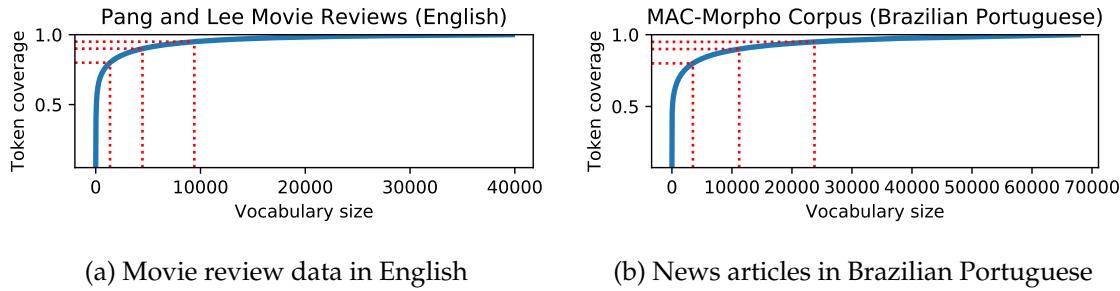


Figure 4.3: Tradeoff between token coverage (y-axis) and vocabulary size, on the `nltk` movie review dataset, after sorting the vocabulary by decreasing frequency. The red dashed lines indicate 80%, 90%, and 95% coverage.

2077 in Figure 4.2: the Lancaster stemmer incorrectly identifies *-ers* as an inflectional suffix of
 2078 *sisters* (by analogy to *fix/fixers*), and both stemmers incorrectly identify *-s* as a suffix of *this*
 2079 and *Williams*. Fortunately, even inaccurate stemming can improve bag-of-words classifi-
 2080 cation models, by merging related strings and thereby reducing the vocabulary size.

2081 Accurately handling irregular orthography requires word-specific rules. **Lemmatizers**
 2082 are systems that identify the underlying lemma of a given wordform. They must avoid the
 2083 over-generalization errors of the stemmers in Figure 4.2, and also handle more complex
 2084 transformations, such as *geese*→*goose*. The output of the WordNet lemmatizer is shown in
 2085 the final line of Figure 4.2. Both stemming and lemmatization are language-specific: an
 2086 English stemmer or lemmatizer is of little use on a text written in another language. The
 2087 discipline of **morphology** relates to the study of word-internal structure, and is described
 2088 in more detail in § 9.1.2.

2089 The value of normalization depends on the data and the task. Normalization re-
 2090 duces the size of the feature space, which can help in generalization. However, there
 2091 is always the risk of merging away linguistically meaningful distinctions. In supervised
 2092 machine learning, regularization and smoothing can play a similar role to normalization
 2093 — preventing the learner from overfitting to rare features — while avoiding the language-
 2094 specific engineering required for accurate normalization. In unsupervised scenarios, such
 2095 as content-based information retrieval (Manning et al., 2008) and topic modeling (Blei
 2096 et al., 2003), normalization is more critical.

2097 4.3.2 How many words?

2098 Limiting the size of the feature vector reduces the memory footprint of the resulting mod-
 2099 els, and increases the speed of prediction. Normalization can help to play this role, but
 2100 a more direct approach is simply to limit the vocabulary to the N most frequent words
 2101 in the dataset. For example, in the `movie-reviews` dataset provided with `nltk` (orig-

2102 inally from Pang et al., 2002), there are 39,768 word types, and 1.58M tokens. As shown
 2103 in Figure 4.3a, the most frequent 4000 word types cover 90% of all tokens, offering an
 2104 order-of-magnitude reduction in the model size. Such ratios are language-specific: in for
 2105 example, in the Brazilian Portuguese Mac-Morpho corpus (Aluísio et al., 2003), attaining
 2106 90% coverage requires more than 10000 word types (Figure 4.3b). This reflects the
 2107 morphological complexity of Portuguese, which includes many more inflectional suffixes
 2108 than English.

2109 Eliminating rare words is not always advantageous for classification performance: for
 2110 example, names, which are typically rare, play a large role in distinguishing topics of news
 2111 articles. Another way to reduce the size of the feature space is to eliminate **stopwords** such
 2112 as *the*, *to*, and *and*, which may seem to play little role in expressing the topic, sentiment,
 2113 or stance. This is typically done by creating a **stoplist** (e.g., `nltk.corpus.stopwords`),
 2114 and then ignoring all terms that match the list. However, corpus linguists and social psy-
 2115 chologists have shown that seemingly inconsequential words can offer surprising insights
 2116 about the author or nature of the text (Biber, 1991; Chung and Pennebaker, 2007). Further-
 2117 more, high-frequency words are unlikely to cause overfitting in discriminative classifiers.
 2118 As with normalization, stopword filtering is more important for unsupervised problems,
 2119 such as term-based document retrieval.

2120 Another alternative for controlling model size is **feature hashing** (Weinberger et al.,
 2121 2009). Each feature is assigned an index using a hash function. If a hash function that
 2122 permits collisions is chosen (typically by taking the hash output modulo some integer),
 2123 then the model can be made arbitrarily small, as multiple features share a single weight.
 2124 Because most features are rare, accuracy is surprisingly robust to such collisions (Ganchev
 2125 and Dredze, 2008).

2126 4.3.3 Count or binary?

2127 Finally, we may consider whether we want our feature vector to include the *count* of each
 2128 word, or its *presence*. This gets at a subtle limitation of linear classification: it's worse to
 2129 have two *failures* than one, but is it really twice as bad? Motivated by this intuition, Pang
 2130 et al. (2002) use binary indicators of presence or absence in the feature vector: $f_j(\mathbf{x}, y) \in$
 2131 $\{0, 1\}$. They find that classifiers trained on these binary vectors tend to outperform feature
 2132 vectors based on word counts. One explanation is that words tend to appear in clumps:
 2133 if a word has appeared once in a document, it is likely to appear again (Church, 2000).
 2134 These subsequent appearances can be attributed to this tendency towards repetition, and
 2135 thus provide little additional information about the class label of the document.

2136 **4.4 Evaluating classifiers**

2137 In any supervised machine learning application, it is critical to reserve a held-out test set.
 2138 This data should be used for only one purpose: to evaluate the overall accuracy of a single
 2139 classifier. Using this data more than once would cause the estimated accuracy to be overly
 2140 optimistic, because the classifier would be customized to this data, and would not perform
 2141 as well as on unseen data in the future. It is usually necessary to set hyperparameters or
 2142 perform feature selection, so you may need to construct a **tuning** or **development set** for
 2143 this purpose, as discussed in § 2.1.5.

2144 There are a number of ways to evaluate classifier performance. The simplest is **accuracy**:
 2145 the number of correct predictions, divided by the total number of instances,

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

2146 Exams are usually graded by accuracy. Why are other metrics necessary? The main
 2147 reason is **class imbalance**. Suppose you are building a classifier to detect whether an
 2148 electronic health record (EHR) describes symptoms of a rare disease, which appears in
 2149 only 1% of all documents in the dataset. A classifier that reports $\hat{y} = \text{NEGATIVE}$ for
 2150 all documents would achieve 99% accuracy, but would be practically useless. We need
 2151 metrics that are capable of detecting the classifier's ability to discriminate between classes,
 2152 even when the distribution is skewed.

2153 One solution is to build a **balanced test set**, in which each possible label is equally rep-
 2154 resented. But in the EHR example, this would mean throwing away 98% of the original
 2155 dataset! Furthermore, the detection threshold itself might be a design consideration: in
 2156 health-related applications, we might prefer a very sensitive classifier, which returned a
 2157 positive prediction if there is even a small chance that $y^{(i)} = \text{POSITIVE}$. In other applica-
 2158 tions, a positive result might trigger a costly action, so we would prefer a classifier that
 2159 only makes positive predictions when absolutely certain. We need additional metrics to
 2160 capture these characteristics.

2161 **4.4.1 Precision, recall, and F-MEASURE**

2162 For any label (e.g., positive for presence of symptoms of a disease), there are two possible
 2163 errors:

- 2164 • **False positive**: the system incorrectly predicts the label.
- 2165 • **False negative**: the system incorrectly fails to predict the label.

2166 Similarly, for any label, there are two ways to be correct:

- 2167 • **True positive:** the system correctly predicts the label.
 2168 • **True negative:** the system correctly predicts that the label does not apply to this
 2169 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**:

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

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

2170 Recall and precision are both conditional likelihoods of a correct prediction, which is why
 2171 their numerators are the same. Recall is conditioned on k being the correct label, $y^{(i)} = k$,
 2172 so the denominator sums over true positive and false negatives. Precision is conditioned
 2173 on k being the prediction, so the denominator sums over true positives and false positives.
 2174 Note that true negatives are not considered in either statistic. The classifier that labels
 2175 every document as “negative” would achieve zero recall; precision would be $\frac{0}{0}$.

2176 Recall and precision are complementary. A high-recall classifier is preferred when
 2177 false positives are cheaper than false negatives: for example, in a preliminary screening
 2178 for symptoms of a disease, the cost of a false positive might be an additional test, while a
 2179 false negative would result in the disease going untreated. Conversely, a high-precision
 2180 classifier is preferred when false positives are more expensive: for example, in spam de-
 2181tection, a false negative is a relatively minor inconvenience, while a false positive might
 2182 mean that an important message goes unread.

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

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

2183 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,

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

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.

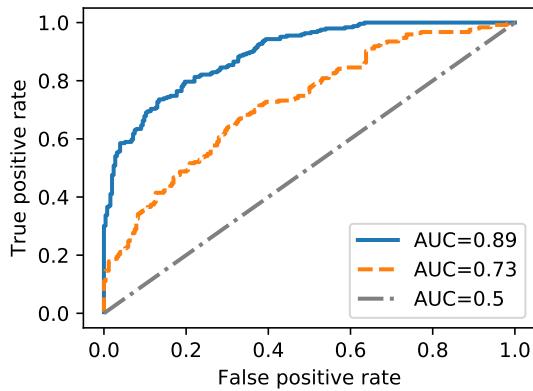


Figure 4.4: ROC curves for three classifiers of varying discriminative power, measured by AUC (area under the curve)

2184 In multi-class problems with unbalanced class distributions, the macro *F*-MEASURE is a
 2185 balanced measure of how well the classifier recognizes each class. In **micro** *F*-MEASURE,
 2186 we compute true positives, false positives, and false negatives for each class, and then add
 2187 them up to compute a single recall, precision, and *F*-MEASURE. This metric is balanced
 2188 across instances rather than classes, so it weights each class in proportion to its frequency
 2189 — unlike macro *F*-MEASURE, which weights each class equally.

2190 4.4.2 Threshold-free metrics

2191 In binary classification problems, it is possible to trade off between recall and precision by
 2192 adding a constant “threshold” to the output of the scoring function. This makes it possible
 2193 to trace out a curve, where each point indicates the performance at a single threshold. In
 2194 the **receiver operating characteristic (ROC)** curve,⁸ the *x*-axis indicates the **false positive**
 2195 **rate**, $\frac{FP}{FP+TN}$, and the *y*-axis indicates the recall, or **true positive rate**. A perfect classifier
 2196 attains perfect recall without any false positives, tracing a “curve” from the origin (0,0) to
 2197 the upper left corner (0,1), and then to (1,1). In expectation, a non-discriminative classifier
 2198 traces a diagonal line from the origin (0,0) to the upper right corner (1,1). Real classifiers
 2199 tend to fall between these two extremes. Examples are shown in Figure 4.4.

2200 The ROC curve can be summarized in a single number by taking its integral, the **area**
 2201 **under the curve (AUC)**. The AUC can be interpreted as the probability that a randomly-
 2202 selected positive example will be assigned a higher score by the classifier than a randomly-

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.

2203 selected negative example. A perfect classifier has $AUC = 1$ (all positive examples score
 2204 higher than all negative examples); a non-discriminative classifier has $AUC = 0.5$ (given
 2205 a randomly selected positive and negative example, either could score higher with equal
 2206 probability); a perfectly wrong classifier would have $AUC = 0$ (all negative examples score
 2207 higher than all positive examples). One advantage of AUC in comparison to F -MEASURE
 2208 is that the baseline rate of 0.5 does not depend on the label distribution.

2209 4.4.3 Classifier comparison and statistical significance

2210 Natural language processing research and engineering often involves comparing different
 2211 classification techniques. In some cases, the comparison is between algorithms, such as
 2212 logistic regression versus averaged perceptron, or L_2 regularization versus L_1 . In other
 2213 cases, the comparison is between feature sets, such as the bag-of-words versus positional
 2214 bag-of-words (see § 4.2.2). **Ablation testing** involves systematically removing (ablating)
 2215 various aspects of the classifier, such as feature groups, and testing the **null hypothesis**
 2216 that the ablated classifier is as good as the full model.

2217 A full treatment of hypothesis testing is beyond the scope of this text, but this section
 2218 contains a brief summary of the techniques necessary to compare classifiers. The main
 2219 aim of hypothesis testing is to determine whether the difference between two statistics
 2220 — for example, the accuracies of two classifiers — is likely to arise by chance. We will
 2221 be concerned with chance fluctuations that arise due to the finite size of the test set.⁹ An
 2222 improvement of 10% on a test set with ten instances may reflect a random fluctuation that
 2223 makes the test set more favorable to classifier c_1 than c_2 ; on another test set with a different
 2224 ten instances, we might find that c_2 does better than c_1 . But if we observe the same 10%
 2225 improvement on a test set with 1000 instances, this is highly unlikely to be explained
 2226 by chance. Such a finding is said to be **statistically significant** at a level p , which is the
 2227 probability of observing an effect of equal or greater magnitude when the null hypothesis
 2228 is true. The notation $p < .05$ indicates that the likelihood of an equal or greater effect is
 2229 less than 5%, assuming the null hypothesis is true.¹⁰

2230 4.4.3.1 The binomial test

2231 The statistical significance of a difference in accuracy can be evaluated using classical tests,
 2232 such as the **binomial test**.¹¹ Suppose that classifiers c_1 and c_2 disagree on N instances in a

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.

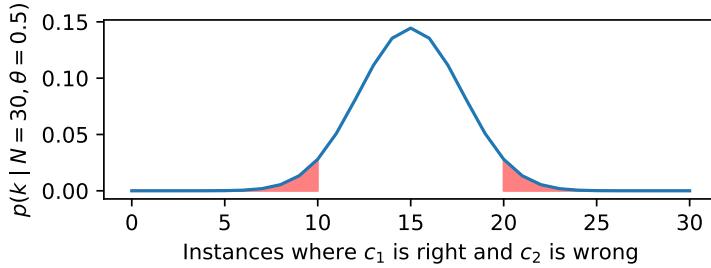


Figure 4.5: Probability mass function for the binomial distribution. The pink highlighted areas represent the cumulative probability for a significance test on an observation of $k = 10$ and $N = 30$.

2233 test set with binary labels, and that c_1 is correct on k of those instances. Under the null hy-
 2234 pothesis that the classifiers are equally accurate, we would expect k/N to be roughly equal
 2235 to $1/2$, and as N increases, k/N should be increasingly close to this expected value. These
 2236 properties are captured by the **binomial distribution**, which is a probability over counts
 2237 of binary random variables. We write $k \sim \text{Binom}(\theta, N)$ to indicate that k is drawn from
 2238 a binomial distribution, with parameter N indicating the number of random “draws”,
 2239 and θ indicating the probability of “success” on each draw. Each draw is an example on
 2240 which the two classifiers disagree, and a “success” is a case in which c_1 is right and c_2 is
 2241 wrong. (The label space is assumed to be binary, so if the classifiers disagree, exactly one
 2242 of them is correct. The test can be generalized to multi-class classification by focusing on
 2243 the examples in which exactly one classifier is correct.)

2244 The **probability mass function** (PMF) of the binomial distribution is,

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

2245 with θ^k representing the probability of the k successes, $(1 - \theta)^{N-k}$ representing the prob-
 2246 ability of the $N - k$ unsuccessful draws. The expression $\binom{N}{k} = \frac{N!}{k!(N-k)!}$ is a binomial
 2247 coefficient, representing the number of possible orderings of events; this ensures that the
 2248 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

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.

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

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

4.4.3.2 *Randomized testing

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

To compare the *F-MEASURE* of two classifiers c_1 and c_2 , we set the function $\delta(\cdot)$ to compute the difference in *F-MEASURE* on the bootstrap sample. If the difference is less than or equal to zero in at least 5% of the samples, then we cannot reject the one-tailed null hypothesis that c_2 is at least as good as c_1 (Berg-Kirkpatrick et al., 2012). We may also be interested in the 95% **confidence interval** around a metric of interest, such as the *F-MEASURE* of a single classifier. This can be computed by sorting the output of Algorithm 7, and then setting the top and bottom of the 95% confidence interval to the values at the 2.5% and 97.5% percentiles of the sorted outputs. Alternatively, you can fit a normal distribution to the set of differences across bootstrap samples, and compute a Gaussian confidence interval from the mean and variance.

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  $\{d^{(t)}\}_{t=1}^M$ 
```

2281 As the number of bootstrap samples goes to infinity, $M \rightarrow \infty$, the bootstrap estimate
 2282 is increasingly accurate. A typical choice for M is 10^4 or 10^5 ; larger numbers of samples
 2283 are necessary for smaller p -values. One way to validate your choice of M is to run the test
 2284 multiple times, and ensure that the p -values are similar; if not, increase M by an order of
 2285 magnitude. This is a heuristic measure of the **variance** of the test, which can decrease
 2286 with the square root \sqrt{M} (Robert and Casella, 2013).

2287 **4.4.4 *Multiple comparisons**

2288 Sometimes it is necessary to perform multiple hypothesis tests, such as when compar-
 2289 ing the performance of several classifiers on multiple datasets. Suppose you have five
 2290 datasets, and you compare four versions of your classifier against a baseline system, for a
 2291 total of 20 comparisons. Even if none of your classifiers is better than the baseline, there
 2292 will be some chance variation in the results, and in expectation you will get one statis-
 2293 tically significant improvement at $p = 0.05 = \frac{1}{20}$. It is therefore necessary to adjust the
 2294 p -values when reporting the results of multiple comparisons.

2295 One approach is to require a threshold of $\frac{\alpha}{m}$ to report a p value of $p < \alpha$ when per-
 2296 forming m tests. This is known as the **Bonferroni correction**, and it limits the overall
 2297 probability of incorrectly rejecting the null hypothesis at α . Another approach is to bound
 2298 the **false discovery rate** (FDR), which is the fraction of null hypothesis rejections that are
 2299 incorrect. Benjamini and Hochberg (1995) propose a p -value correction that bounds the
 2300 fraction of false discoveries at α : sort the p -values of each individual test in ascending
 2301 order, and set the significance threshold equal to largest k such that $p_k \leq \frac{k}{m}\alpha$. If $k > 1$, the
 2302 FDR adjustment is more permissive than the Bonferroni correction.

2303 4.5 Building datasets

2304 Sometimes, if you want to build a classifier, you must first build a dataset of your own.
2305 This includes selecting a set of documents or instances to annotate, and then performing
2306 the annotations. The scope of the dataset may be determined by the application: if you
2307 want to build a system to classify electronic health records, then you must work with a
2308 corpus of records of the type that your classifier will encounter when deployed. In other
2309 cases, the goal is to build a system that will work across a broad range of documents. In
2310 this case, it is best to have a *balanced* corpus, with contributions from many styles and
2311 genres. For example, the Brown corpus draws from texts ranging from government doc-
2312 uments to romance novels (Francis, 1964), and the Google Web Treebank includes an-
2313 notations for five “domains” of web documents: question answers, emails, newsgroups,
2314 reviews, and blogs (Petrov and McDonald, 2012).

2315 4.5.1 Metadata as labels

2316 Annotation is difficult and time-consuming, and most people would rather avoid it. It
2317 is sometimes possible to exploit existing metadata to obtain labels for training a classi-
2318 fier. For example, reviews are often accompanied by a numerical rating, which can be
2319 converted into a classification label (see § 4.1). Similarly, the nationalities of social media
2320 users can be estimated from their profiles (Dredze et al., 2013) or even the time zones of
2321 their posts (Gouws et al., 2011). More ambitiously, we may try to classify the political af-
2322 filiations of social media profiles based on their social network connections to politicians
2323 and major political parties (Rao et al., 2010).

2324 The convenience of quickly constructing large labeled datasets without manual an-
2325 notation is appealing. However this approach relies on the assumption that unlabeled
2326 instances — for which metadata is unavailable — will be similar to labeled instances.
2327 Consider the example of labeling the political affiliation of social media users based on
2328 their network ties to politicians. If a classifier attains high accuracy on such a test set,
2329 is it safe to assume that it accurately predicts the political affiliation of all social media
2330 users? Probably not. Social media users who establish social network ties to politicians
2331 may be more likely to mention politics in the text of their messages, as compared to the
2332 average user, for whom no political metadata is available. If so, the accuracy on a test set
2333 constructed from social network metadata would give an overly optimistic picture of the
2334 method’s true performance on unlabeled data.

2335 4.5.2 Labeling data

2336 In many cases, there is no way to get ground truth labels other than manual annotation.
2337 An annotation protocol should satisfy several criteria: the annotations should be *expressive*
2338 enough to capture the phenomenon of interest; they should be *replicable*, meaning that

2339 another annotator or team of annotators would produce very similar annotations if given
2340 the same data; and they should be *scalable*, so that they can be produced relatively quickly.
2341 Hovy and Lavid (2010) propose a structured procedure for obtaining annotations that
2342 meet these criteria, which is summarized below.

- 2343 1. **Determine what the annotations are to include.** This is usually based on some
2344 theory of the underlying phenomenon: for example, if the goal is to produce an-
2345 notations about the emotional state of a document’s author, one should start with a
2346 theoretical account of the types or dimensions of emotion (e.g., Mohammad and Tur-
2347 ney, 2013). At this stage, the tradeoff between expressiveness and scalability should
2348 be considered: a full instantiation of the underlying theory might be too costly to
2349 annotate at scale, so reasonable approximations should be considered.
- 2350 2. Optionally, one may **design or select a software tool to support the annotation**
2351 **effort.** Existing general-purpose annotation tools include BRAT (Stenetorp et al.,
2352 2012) and MMAX2 (Müller and Strube, 2006).
- 2353 3. **Formalize the instructions for the annotation task.** To the extent that the instruc-
2354 tions are not explicit, the resulting annotations will depend on the intuitions of the
2355 annotators. These intuitions may not be shared by other annotators, or by the users
2356 of the annotated data. Therefore explicit instructions are critical to ensuring the an-
2357 notations are replicable and usable by other researchers.
- 2358 4. **Perform a pilot annotation** of a small subset of data, with multiple annotators for
2359 each instance. This will give a preliminary assessment of both the replicability and
2360 scalability of the current annotation instructions. Metrics for computing the rate of
2361 agreement are described below. Manual analysis of specific disagreements should
2362 help to clarify the instructions, and may lead to modifications of the annotation task
2363 itself. For example, if two labels are commonly conflated by annotators, it may be
2364 best to merge them.
- 2365 5. **Annotate the data.** After finalizing the annotation protocol and instructions, the
2366 main annotation effort can begin. Some, if not all, of the instances should receive
2367 multiple annotations, so that inter-annotator agreement can be computed. In some
2368 annotation projects, instances receive many annotations, which are then aggregated
2369 into a “consensus” label (e.g., Danescu-Niculescu-Mizil et al., 2013). However, if the
2370 annotations are time-consuming or require significant expertise, it may be preferable
2371 to maximize scalability by obtaining multiple annotations for only a small subset of
2372 examples.
- 2373 6. **Compute and report inter-annotator agreement, and release the data.** In some
2374 cases, the raw text data cannot be released, due to concerns related to copyright or

privacy. In these cases, one solution is to publicly release **stand-off annotations**, which contain links to document identifiers. The documents themselves can be released under the terms of a licensing agreement, which can impose conditions on how the data is used. It is important to think through the potential consequences of releasing data: people may make personal data publicly available without realizing that it could be redistributed in a dataset and publicized far beyond their expectations (boyd and Crawford, 2012).

2382 4.5.2.1 Measuring inter-annotator agreement

2383 To measure the replicability of annotations, a standard practice is to compute the extent to
 2384 which annotators agree with each other. If the annotators frequently disagree, this casts
 2385 doubt on either their reliability or on the annotation system itself. For classification, one
 2386 can compute the frequency with which the annotators agree; for rating scales, one can
 2387 compute the average distance between ratings. These raw agreement statistics must then
 2388 be compared with the rate of **chance agreement** — the level of agreement that would be
 2389 obtained between two annotators who ignored the data.

2390 **Cohen's Kappa** is widely used for quantifying the agreement on discrete labeling
 2391 tasks (Cohen, 1960; Carletta, 1996),¹²

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

2392 The numerator is the difference between the observed agreement and the chance agree-
 2393 ment, and the denominator is the difference between perfect agreement and chance agree-
 2394 ment. Thus, $\kappa = 1$ when the annotators agree in every case, and $\kappa = 0$ when the annota-
 2395 tors agree only as often as would happen by chance. Various heuristic scales have been
 2396 proposed for determining when κ indicates "moderate", "good", or "substantial" agree-
 2397 ment; for reference, Lee and Narayanan (2005) report $\kappa \approx 0.45 - 0.47$ for annotations
 2398 of emotions in spoken dialogues, which they describe as "moderate agreement"; Stolcke
 2399 et al. (2000) report $\kappa = 0.8$ for annotations of **dialogue acts**, which are labels for the pur-
 2400 pose of each turn in a conversation.

2401 When there are two annotators, the expected chance agreement is computed as,

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

2402 where k is a sum over labels, and $\hat{\Pr}(Y = k)$ is the empirical probability of label k across
 2403 all annotations. The formula is derived from the expected number of agreements if the
 2404 annotations were randomly shuffled. Thus, in a binary labeling task, if one label is applied
 2405 to 90% of instances, chance agreement is $.9^2 + .1^2 = .82$.

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

2406 **4.5.2.2 Crowdsourcing**

2407 Crowdsourcing is often used to rapidly obtain annotations for classification problems.
 2408 For example, **Amazon Mechanical Turk** makes it possible to define “human intelligence
 2409 tasks (hits)”, such as labeling data. The researcher sets a price for each set of annotations
 2410 and a list of minimal qualifications for annotators, such as their native language and their
 2411 satisfaction rate on previous tasks. The use of relatively untrained “crowdworkers” con-
 2412 trasts with earlier annotation efforts, which relied on professional linguists (Marcus et al.,
 2413 1993). However, crowdsourcing has been found to produce reliable annotations for many
 2414 language-related tasks (Snow et al., 2008). Crowdsourcing is part of the broader field of
 2415 **human computation** (Law and Ahn, 2011).

2416 **Additional resources**

2417 Many of the preprocessing issues discussed in this chapter also arise in information re-
 2418 trieval. See (Manning et al., 2008, chapter 2) for discussion of tokenization and related
 2419 algorithms.

2420 **Exercises**

- 2421 1. As noted in § 4.3.3, words tend to appear in clumps, with subsequent occurrences
 2422 of a word being more probable. More concretely, if word j has probability $\phi_{y,j}$
 2423 of appearing in a document with label y , then the probability of two appearances
 2424 ($x_j^{(i)} = 2$) is greater than $\phi_{y,j}^2$.

2425 Suppose you are applying Naïve Bayes to a binary classification. Focus on a word j
 2426 which is more probable under label $y = 1$, so that,

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

2427 Now suppose that $x_j^{(i)} > 1$. All else equal, will the classifier overestimate or under-
 2428 estimate the posterior $\Pr(y = 1 \mid x)$?

- 2429 2. Prove that F-measure is never greater than the arithmetic mean of recall and pre-
 2430 cision, $\frac{r+p}{2}$. Your solution should also show that F-measure is equal to $\frac{r+p}{2}$ iff $r = p$.
- 2431 3. Given a binary classification problem in which the probability of the “positive” label
 2432 is equal to α , what is the expected F-MEASURE of a random classifier which ignores
 2433 the data, and selects $\hat{y} = +1$ with probability $\frac{1}{2}$? (Assume that $p(\hat{y}) \perp p(y)$.) What is
 2434 the expected F-MEASURE of a classifier that selects $\hat{y} = +1$ with probability α (also
 2435 independent of $y^{(i)}$)? Depending on α , which random classifier will score better?

- 2436 4. Suppose that binary classifiers c_1 and c_2 disagree on $N = 30$ cases, and that c_1 is
 2437 correct in $k = 10$ of those cases.
- 2438 • Write a program that uses primitive functions such as `exp` and `factorial` to com-
 2439 pute the **two-tailed** p -value — you may use an implementation of the “choose”
 2440 function if one is available. Verify your code against the output of a library for
 2441 computing the binomial test or the binomial CDF, such as `scipy.stats.binom`
 2442 in Python.
- 2443 • Then use a randomized test to try to obtain the same p -value. In each sample,
 2444 draw from a binomial distribution with $N = 30$ and $\theta = \frac{1}{2}$. Count the fraction
 2445 of samples in which $k \leq 10$. This is the one-tailed p -value; double this to
 2446 compute the two-tailed p -value.
- 2447 • Try this with varying numbers of bootstrap samples: $M \in \{100, 1000, 5000, 10000\}$.
 2448 For $M = 100$ and $M = 1000$, run the test 10 times, and plot the resulting p -
 2449 values.
- 2450 • Finally, perform the same tests for $N = 70$ and $k = 25$.
- 2451 5. SemCor 3.0 is a labeled dataset for word sense disambiguation. You can download
 2452 it,¹³ or access it in `nltk.corpora.semcor`.
- 2453 Choose a word that appears at least ten times in SemCor (*find*), and annotate its
 2454 WordNet senses across ten randomly-selected examples, without looking at the ground
 2455 truth. Use online WordNet to understand the definition of each of the senses.¹⁴ Have
 2456 a partner do the same annotations, and compute the raw rate of agreement, expected
 2457 chance rate of agreement, and Cohen’s kappa.
- 2458 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
 2459 randomly-selected 400 reviews as a test set.
- 2460 Download a sentiment lexicon, such as the one currently available from Bing Liu,
 2461 <https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html>. Tokenize
 2462 the data, and classify each document as positive iff it has more positive sentiment
 2463 words than negative sentiment words. Compute the accuracy and *F*-MEASURE on
 2464 detecting positive reviews on the test set, using this lexicon-based classifier.
- 2465 Then train a discriminative classifier (averaged perceptron or logistic regression) on
 2466 the training set, and compute its accuracy and *F*-MEASURE on the test set.
- 2467 Determine whether the differences are statistically significant, using two-tailed hy-
 2468 pothesis tests: Binomial for the difference in accuracy, and bootstrap for the differ-
 2469 ence in macro-*F*-MEASURE.

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>

2471 The remaining problems will require you to build a classifier and test its properties. Pick
2472 a multi-class text classification dataset that is not already tokenized. One example is a
2473 dataset of twelve categories of complaints from the U.S. Consumer Financial Protection
2474 Bureau.¹⁵ Divide your data into training (60%), development (20%), and test sets (20%), if
2475 no such division already exists. If your dataset is very large, you may want to focus on a
2476 few thousand instances at first.

2477 7. Compare various vocabulary sizes of $10^2, 10^3, 10^4, 10^5$, using the most frequent words
2478 in each case (you may use any reasonable tokenizer). Train logistic regression clas-
2479 sifiers for each vocabulary size, and apply them to the development set. Plot the
2480 accuracy and Macro-*F*-MEASURE with the increasing vocabulary size. For each vo-
2481 cabulary size, tune the regularizer to maximize accuracy on a subset of data that is
2482 held out from the training set.

2483 8. Compare the following tokenization algorithms:

- 2484 • Whitespace, using a regular expression
2485 • Penn Treebank
2486 • Split input into five-character units, regardless of whitespace or punctuation

2487 Compute the token/type ratio for each tokenizer on the training data, and explain
2488 what you find. Train your classifier on each tokenized dataset, tuning the regularizer
2489 on a subset of data that is held out from the training data. Tokenize the development
2490 set, and report accuracy and Macro-*F*-MEASURE.

2491 9. Apply the Porter and Lancaster stemmers to the training set, using any reasonable
2492 tokenizer, and compute the token/type ratios. Train your classifier on the stemmed
2493 data, and compute the accuracy and Macro-*F*-MEASURE on stemmed development
2494 data, again using a held-out portion of the training data to tune the regularizer.

2495 10. Identify the best combination of vocabulary filtering, tokenization, and stemming
2496 from the previous three problems. Apply this preprocessing to the test set, and
2497 compute the test set accuracy and Macro-*F*-MEASURE. Compare against a baseline
2498 system that applies no vocabulary filtering, whitespace tokenization, and no stem-
2499 ming.

2500 Use the binomial test to determine whether your best-performing system is signifi-
2501 cantly more accurate than the baseline.

2502 Use the bootstrap test with $M = 10^4$ to determine whether your best-performing
2503 system achieves significantly higher macro-*F*-MEASURE.

¹⁵<https://catalog.data.gov/dataset/consumer-complaint-database>, retrieved September 12, 2018.

2504

Chapter 5

2505

Learning without supervision

2506 So far we've assumed the following setup:

- 2507 • a **training set** where you get observations x and labels y ;
2508 • a **test set** where you only get observations x .

2509 Without labeled data, is it possible to learn anything? This scenario is known as **unsu-**
2510 **pervised learning**, and we will see that indeed it is possible to learn about the underlying
2511 structure of unlabeled observations. This chapter will also explore some related scenarios:
2512 **semi-supervised learning**, in which only some instances are labeled, and **domain adap-**
2513 **tation**, in which the training data differs from the data on which the trained system will
2514 be deployed.

2515

5.1 Unsupervised learning

2516 To motivate unsupervised learning, consider the problem of word sense disambiguation
2517 (\S 4.2). The goal is to classify each instance of a word, such as *bank* into a sense,

- 2518 • bank#1: a financial institution
2519 • bank#2: the land bordering a river

2520 It is difficult to obtain sufficient training data for word sense disambiguation, because
2521 even a large corpus will contain only a few instances of all but the most common words.
2522 Is it possible to learn anything about these different senses without labeled data?

2523 Word sense disambiguation is usually performed using feature vectors constructed
2524 from the local context of the word to be disambiguated. For example, for the word

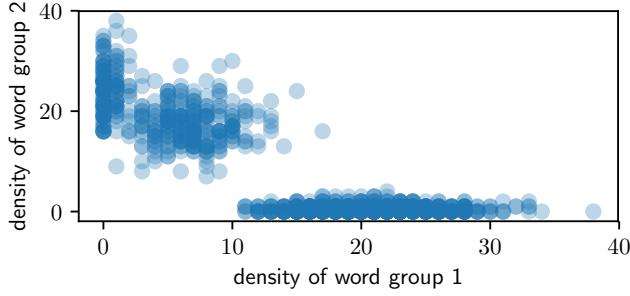


Figure 5.1: Counts of words from two different context groups

2525 *bank*, the immediate context might typically include words from one of the following two
 2526 groups:

- 2527 1. *financial, deposits, credit, lending, capital, markets, regulated, reserve, liquid, assets*
 2528 2. *land, water, geography, stream, river, flow, deposits, discharge, channel, ecology*

2529 Now consider a scatterplot, in which each point is a document containing the word *bank*.
 2530 The location of the document on the x -axis is the count of words in group 1, and the
 2531 location on the y -axis is the count for group 2. In such a plot, shown in Figure 5.1, two
 2532 “blobs” might emerge, and these blobs correspond to the different senses of *bank*.

2533 Here’s a related scenario, from a different problem. Suppose you download thousands
 2534 of news articles, and make a scatterplot, where each point corresponds to a document:
 2535 the x -axis is the frequency of the group of words (*hurricane, winds, storm*); the y -axis is the
 2536 frequency of the group (*election, voters, vote*). This time, three blobs might emerge: one
 2537 for documents that are largely about a hurricane, another for documents largely about a
 2538 election, and a third for documents about neither topic.

2539 These clumps represent the underlying structure of the data. But the two-dimensional
 2540 scatter plots are based on groupings of context words, and in real scenarios these word
 2541 lists are unknown. Unsupervised learning applies the same basic idea, but in a high-
 2542 dimensional space with one dimension for every context word. This space can’t be di-
 2543 rectly visualized, but the idea is the same: try to identify the underlying structure of the
 2544 observed data, such that there are a few clusters of points, each of which is internally
 2545 coherent. **Clustering** algorithms are capable of finding such structure automatically.

2546 5.1.1 **K-means** clustering

2547 Clustering algorithms assign each data point to a discrete cluster, $z_i \in 1, 2, \dots, K$. One of
 2548 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                       $\triangleright$  reassign 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

```

2549 a cluster assignment for each instance, and a central (“mean”) location for each cluster.
 2550 K -means iterates between updates to the assignments and the centers:

- 2551 1. each instance is placed in the cluster with the closest center;
 2552 2. each center is recomputed as the average over points in the cluster.

2553 This is formalized in Algorithm 8. The term $\|\mathbf{x}^{(i)} - \boldsymbol{\nu}\|^2$ refers to the squared Euclidean
 2554 norm, $\sum_{j=1}^V (x_j^{(i)} - \nu_j)^2$.

2555 **Soft K -means** is a particularly relevant variant. Instead of directly assigning each
 2556 point to a specific cluster, soft K -means assigns to each point a *distribution* over clusters
 2557 $\mathbf{q}^{(i)}$, 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
 2558 the distance of $\mathbf{x}^{(i)}$ to the cluster center $\boldsymbol{\nu}_k$. In turn, the center of each cluster is computed
 2559 from a weighted average of the points in the cluster,

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

2560 We will now explore a probabilistic version of soft K -means clustering, based on **expectation-**
 2561 **maximization** (EM). Because EM clustering can be derived as an approximation to maximum-
 2562 likelihood estimation, it can be extended in a number of useful ways.

2563 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** z , we obtain the marginal probability of the observed instances \mathbf{x} :

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

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

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

2564 The parameters $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$ can be obtained by maximizing the marginal likelihood in
 2565 Equation 5.5. Why is this the right thing to maximize? Without labels, discriminative
 2566 learning is impossible — there's nothing to discriminate. So maximum likelihood is all
 2567 we have.

2568 When the labels are observed, we can estimate the parameters of the Naïve Bayes
 2569 probability model separately for each label. But marginalizing over the labels couples
 2570 these parameters, making direct optimization of $\log p(\mathbf{x})$ intractable. We will approxi-
 2571 mate the log-likelihood by introducing an auxiliary variable $\mathbf{q}^{(i)}$, which is a distribution
 2572 over the label set $\mathcal{Z} = \{1, 2, \dots, K\}$. The optimization procedure will alternate between
 2573 updates to \mathbf{q} and updates to the parameters $(\boldsymbol{\phi}, \boldsymbol{\mu})$. Thus, $\mathbf{q}^{(i)}$ plays here as in soft K -
 2574 means.

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

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

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

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

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

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.

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

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

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

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,

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

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

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

where C is a constant with respect to $\boldsymbol{\phi}$ — see Equation 2.12 in § 2.1 for more discussion 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,

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

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

This update sets ϕ_z equal to the relative frequency estimate of the *expected counts* under the distribution q . As in supervised Naïve Bayes, we can smooth these counts by adding 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 expected frequency of cluster z . These probabilities can also be smoothed. In sum, the M-step is just like Naïve Bayes, but with expected counts rather than observed counts.

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

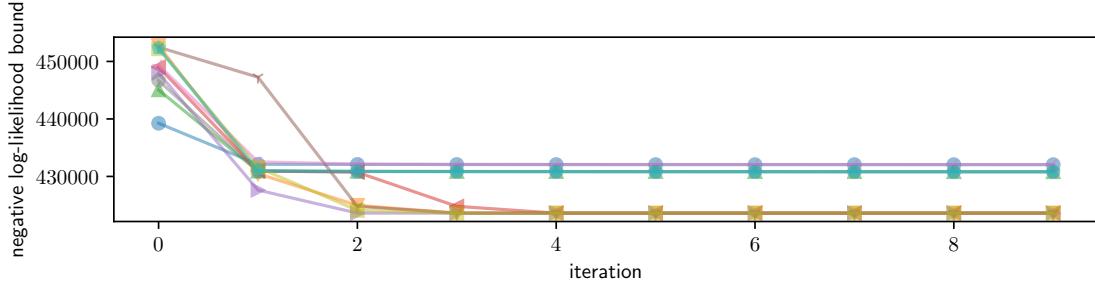


Figure 5.2: Sensitivity of expectation-maximization to initialization. Each line shows the progress of optimization from a different random initialization.

2609 5.1.3 EM as an optimization algorithm

2610 Algorithms that alternate between updating subsets of the parameters are called **coordi-**
 2611 **nate ascent** algorithms. The objective J (the lower bound on the marginal likelihood of
 2612 the data) is separately convex in q and (μ, ϕ) , but it is not jointly convex in all terms; this
 2613 condition is known as **biconvexity**. Each step of the expectation-maximization algorithm
 2614 is guaranteed not to decrease the lower bound J , which means that EM will converge
 2615 towards a solution at which no nearby points yield further improvements. This solution
 2616 is a **local optimum** — it is as good or better than any of its immediate neighbors, but is
 2617 *not* guaranteed to be optimal among all possible configurations of (q, μ, ϕ) .

2618 The fact that there is no guarantee of global optimality means that initialization is
 2619 important: where you start can determine where you finish. To illustrate this point,
 2620 Figure 5.2 shows the objective function for EM with ten different random initializations:
 2621 while the objective function improves monotonically in each run, it converges to several
 2622 different values.¹ For the convex objectives that we encountered in chapter 2, it was not
 2623 necessary to worry about initialization, because gradient-based optimization guaranteed
 2624 to reach the global minimum. But in expectation-maximization — as in the deep neural
 2625 networks from chapter 3 — initialization matters.

2626 In **hard EM**, each $q^{(i)}$ distribution assigns probability of 1 to a single label $\hat{z}^{(i)}$, and zero
 2627 probability to all others (Neal and Hinton, 1998). This is similar in spirit to K -means clus-
 2628 tering, and can outperform standard EM in some cases (Spitkovsky et al., 2010). Another
 2629 variant of expectation-maximization incorporates stochastic gradient descent (SGD): after
 2630 performing a local E-step at each instance $x^{(i)}$, we immediately make a gradient update
 2631 to the parameters (μ, ϕ) . This algorithm has been called **incremental expectation maxi-**
 2632 **mization** (Neal and Hinton, 1998) and **online expectation maximization** (Sato and Ishii,
 2633 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.

2634 optimum for the likelihood $p(\mathbf{x} \mid z)$, and in online settings where new data is constantly
 2635 streamed in (see Liang and Klein, 2009, for a comparison for online EM variants).

2636 **5.1.4 How many clusters?**

2637 So far, we have assumed that the number of clusters K is given. In some cases, this as-
 2638 sumption is valid. For example, a lexical semantic resource like WordNet might define the
 2639 number of senses for a word. In other cases, the number of clusters could be a parameter
 2640 for the user to tune: some readers want a coarse-grained clustering of news stories into
 2641 three or four clusters, while others want a fine-grained clustering into twenty or more.
 2642 But many times there is little extrinsic guidance for how to choose K .

2643 One solution is to choose the number of clusters to maximize a metric of clustering
 2644 quality. The other parameters μ and ϕ are chosen to maximize the log-likelihood bound
 2645 J , so this might seem a potential candidate for tuning K . However, J will never decrease
 2646 with K : if it is possible to obtain a bound of J_K with K clusters, then it is always possible
 2647 to do at least as well with $K + 1$ clusters, by simply ignoring the additional cluster and
 2648 setting its probability to zero in q and μ . It is therefore necessary to introduce a penalty
 2649 for model complexity, so that fewer clusters are preferred. For example, the Akaike Infor-
 2650 mation Crition (AIC; Akaike, 1974) is the linear combination of the number of parameters
 2651 and the log-likelihood,

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

2652 where M is the number of parameters. In an expectation-maximization clustering algo-
 2653 rithm, $M = K \times V + K$. Since the number of parameters increases with the number of
 2654 clusters K , the AIC may prefer more parsimonious models, even if they do not fit the data
 2655 quite as well.

2656 Another choice is to maximize the **predictive likelihood** on heldout data. This data
 2657 is not used to estimate the model parameters ϕ and μ , and so it is not the case that the
 2658 likelihood on this data is guaranteed to increase with K . Figure 5.3 shows the negative
 2659 log-likelihood on training and heldout data, as well as the AIC.

2660 ***Bayesian nonparametrics** An alternative approach is to treat the number of clusters
 2661 as another latent variable. This requires statistical inference over a set of models with a
 2662 variable number of clusters. This is not possible within the framework of expecta-
 2663 tion maximization, but there are several alternative inference procedures which can be ap-
 2664 plied, including **Markov Chain Monte Carlo (MCMC)**, which is briefly discussed in
 2665 § 5.5 (for more details, see Chapter 25 of Murphy, 2012). Bayesian nonparametrics have
 2666 been applied to the problem of unsupervised word sense induction, learning not only the
 2667 word senses but also the number of senses per word (Reisinger and Mooney, 2010).

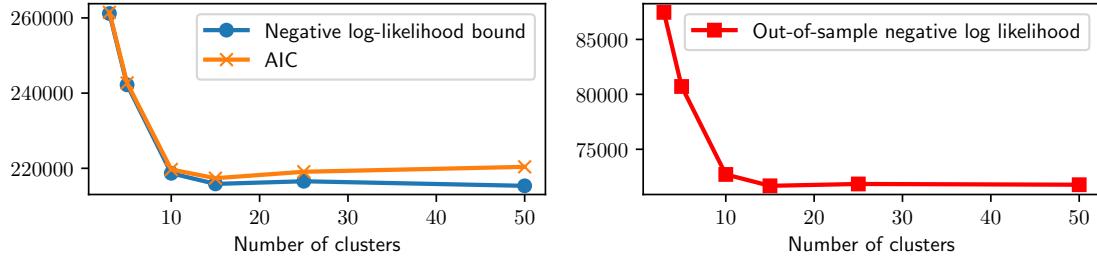


Figure 5.3: The negative log-likelihood and AIC for several runs of expectation-maximization, on synthetic data. Although the data was generated from a model with $K = 10$, the optimal number of clusters is $\hat{K} = 15$, according to AIC and the heldout log-likelihood. The training set log-likelihood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

2668 5.2 Applications of expectation-maximization

2669 EM is not really an “algorithm” like, say, quicksort. Rather, it is a framework for learning
2670 with missing data. The recipe for using EM on a problem of interest is:

- 2671 • Introduce latent variables z , such that it is easy to write the probability $P(\mathbf{x}, z)$. It
2672 should also be easy to estimate the associated parameters, given knowledge of z .
- 2673 • Derive the E-step updates for $q(z)$, which is typically factored as $q(z) = \prod_{i=1}^N q_{z^{(i)}}(z^{(i)})$,
2674 where i is an index over instances.
- 2675 • The M-step updates typically correspond to the soft version of a probabilistic super-
2676 vised learning algorithm, like Naïve Bayes.

2677 This section discusses a few of the many applications of this general framework.

2678 5.2.1 Word sense induction

2679 The chapter began by considering the problem of word sense disambiguation when the
2680 senses are not known in advance. Expectation-maximization can be applied to this prob-
2681 lem by treating each cluster as a word sense. Each instance represents the use of an
2682 ambiguous word, and $\mathbf{x}^{(i)}$ is a vector of counts for the other words that appear nearby:
2683 Schütze (1998) uses all words within a 50-word window. The probability $p(\mathbf{x}^{(i)} | z)$ can be
2684 set to the multinomial distribution, as in Naïve Bayes. The EM algorithm can be applied
2685 directly to this data, yielding clusters that (hopefully) correspond to the word senses.

Better performance can be obtained by first applying **singular value decomposition** (SVD) to the matrix of context-counts $\mathbf{C}_{ij} = \text{count}(i, j)$, 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 annotated examples, ensuring that each label y corresponds to the desired concept. By adding unlabeled examples, it is possible cover a greater fraction of the features than would appear in 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 $x^{(i)} | y^{(i)}, z^{(i)} \sim \text{Multinomial}(\phi_{z^{(i)}})$.

2705 The left sum is identical to the objective in Naïve Bayes; the right sum is the marginal log-
 2706 likelihood for expectation-maximization clustering, from Equation 5.5. We can construct a
 2707 lower bound on this log-likelihood by introducing distributions $q^{(j)}$ for all $j \in \{N_\ell + 1, \dots, N_\ell + N_u\}$.
 2708 The E-step updates these distributions; the M-step updates the parameters ϕ and μ , us-
 2709 ing the expected counts from the unlabeled data and the observed counts from the labeled
 2710 data.

2711 A critical issue in semi-supervised learning is how to balance the impact of the labeled
 2712 and unlabeled data on the classifier weights, especially when the unlabeled data is much
 2713 larger than the labeled dataset. The risk is that the unlabeled data will dominate, caus-
 2714 ing the parameters to drift towards a “natural clustering” of the instances — which may
 2715 not correspond to a good classifier for the labeled data. One solution is to heuristically
 2716 reweight the two components of Equation 5.26, tuning the weight of the two components
 2717 on a heldout development set (Nigam et al., 2000).

2718 **5.2.3 Multi-component modeling**

2719 As a final application, let’s return to fully supervised classification. A classic dataset for
 2720 text classification is 20 newsgroups, which contains posts to a set of online forums, called
 2721 newsgroups. One of the newsgroups is `comp.sys.mac.hardware`, which discusses Ap-
 2722 ple computing hardware. Suppose that within this newsgroup there are two kinds of
 2723 posts: reviews of new hardware, and question-answer posts about hardware problems.
 2724 The language in these *components* of the `mac.hardware` class might have little in com-
 2725 mon; if so, it would be better to model these components separately, rather than treating
 2726 their union as a single class. However, the component responsible for each instance is not
 2727 directly observed.

2728 Recall that Naïve Bayes is based on a generative process, which provides a stochastic
 2729 explanation for the observed data. In Naïve Bayes, each label is drawn from a categorical
 2730 distribution with parameter μ , and each vector of word counts is drawn from a multi-
 2731 nomial distribution with parameter ϕ_y . For multi-component modeling, we envision a
 2732 slightly different generative process, incorporating both the observed label $y^{(i)}$ and the
 2733 latent component $z^{(i)}$. This generative process is shown in Algorithm 9. A new parameter
 2734 $\beta_{y^{(i)}}$ defines the distribution of components, conditioned on the label $y^{(i)}$. The component,
 2735 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,

$$\beta_{y,z} = \frac{E_q [\text{count}(y, z)]}{\sum_{z'=1}^{K_z} E_q [\text{count}(y, z')]} \quad [5.30]$$

$$\phi_{z,j} = \frac{E_q [\text{count}(z, j)]}{\sum_{j'=1}^V E_q [\text{count}(z, j')]} \quad [5.31]$$

²⁷³⁶ **5.3 Semi-supervised learning**

²⁷³⁷ In semi-supervised learning, the learner makes use of both labeled and unlabeled data.
²⁷³⁸ 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

2775 **Co-training** is an iterative multi-view learning algorithm, in which there are separate
 2776 classifiers for each view (Blum and Mitchell, 1998). At each iteration of the algorithm, each
 2777 classifier predicts labels for a subset of the unlabeled instances, using only the features
 2778 available in its view. These predictions are then used as ground truth to train the classifiers
 2779 associated with the other views. In the example shown in Table 5.2, the classifier on $\mathbf{x}^{(1)}$
 2780 might correctly label instance #5 as a person, because of the feature *Dr*; this instance would
 2781 then serve as training data for the classifier on $\mathbf{x}^{(2)}$, which would then be able to correctly
 2782 label instance #6, thanks to the feature *recommended*. If the views are truly independent,
 2783 this procedure is robust to drift. Furthermore, it imposes no restrictions on the classifiers
 2784 that can be used for each view.

2785 Word-sense disambiguation is particularly suited to multi-view learning, thanks to the
 2786 heuristic of “one sense per discourse”: if a polysemous word is used more than once in
 2787 a given text or conversation, all usages refer to the same sense (Gale et al., 1992). This
 2788 motivates a multi-view learning approach, in which one view corresponds to the local
 2789 context (the surrounding words), and another view corresponds to the global context at
 2790 the document level (Yarowsky, 1995). The local context view is first trained on a small
 2791 seed dataset. We then identify its most confident predictions on unlabeled instances. The
 2792 global context view is then used to extend these confident predictions to other instances
 2793 within the same documents. These new instances are added to the training data to the
 2794 local context classifier, which is retrained and then applied to the remaining unlabeled
 2795 data.

2796 5.3.2 Graph-based algorithms

2797 Another family of approaches to semi-supervised learning begins by constructing a graph,
 2798 in which pairs of instances are linked with symmetric weights $\omega_{i,j}$, e.g.,

$$\omega_{i,j} = \exp(-\alpha \times \|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2). \quad [5.32]$$

2799 The goal is to use this weighted graph to propagate labels from a small set of labeled
 2800 instances to larger set of unlabeled instances.

2801 In **label propagation**, this is done through a series of matrix operations (Zhu et al.,
 2802 2003). Let \mathbf{Q} be a matrix of size $N \times K$, in which each row $\mathbf{q}^{(i)}$ describes the labeling
 2803 of instance i . When ground truth labels are available, then $\mathbf{q}^{(i)}$ is an indicator vector,
 2804 with $q_{y^{(i)}}^{(i)} = 1$ and $q_{y' \neq y^{(i)}}^{(i)} = 0$. Let us refer to the submatrix of rows containing labeled
 2805 instances as \mathbf{Q}_L , and the remaining rows as \mathbf{Q}_U . The rows of \mathbf{Q}_U are initialized to assign
 2806 equal probabilities to all labels, $q_{i,k} = \frac{1}{K}$.

2807 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 manip-
 ulate; 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,

$$\tilde{\mathbf{Q}}_U \leftarrow \mathbf{T}\mathbf{Q} \quad [5.34]$$

$$\mathbf{s} \leftarrow \tilde{\mathbf{Q}}_U \mathbf{1} \quad [5.35]$$

$$\mathbf{Q}_U \leftarrow \text{Diag}(\mathbf{s})^{-1} \tilde{\mathbf{Q}}_U. \quad [5.36]$$

2808 The expression $\tilde{\mathbf{Q}}_U \mathbf{1}$ indicates multiplication of $\tilde{\mathbf{Q}}_U$ by a column vector of ones, which is
 2809 equivalent to computing the sum of each row of $\tilde{\mathbf{Q}}_U$. The matrix $\text{Diag}(\mathbf{s})$ is a diagonal
 2810 matrix with the elements of \mathbf{s} on the diagonals. The product $\text{Diag}(\mathbf{s})^{-1} \tilde{\mathbf{Q}}_U$ has the effect
 2811 of normalizing the rows of $\tilde{\mathbf{Q}}_U$, so that each row of \mathbf{Q}_U is a probability distribution over
 2812 labels.

2813 5.4 Domain adaptation

2814 In many practical scenarios, the labeled data differs in some key respect from the data
 2815 to which the trained model is to be applied. A classic example is in consumer reviews:
 2816 we may have labeled reviews of movies (the source domain), but we want to predict the
 2817 reviews of appliances (the target domain). A similar issues arise with genre differences:
 2818 most linguistically-annotated data is news text, but application domains range from social
 2819 media to electronic health records. In general, there may be several source and target
 2820 domains, each with their own properties; however, for simplicity, this discussion will
 2821 focus mainly on the case of a single source and target domain.

2822 The simplest approach is “direct transfer”: train a classifier on the source domain,
 2823 and apply it directly to the target domain. The accuracy of this approach depends on the
 2824 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, e.g.,

$$\hat{y} = \operatorname{argmax}_y \lambda_s \Psi_s(\mathbf{x}, y) + (1 - \lambda_s) \Psi_t(\mathbf{x}, y), \quad [5.37]$$

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,

$$\hat{y}_S = \operatorname{argmax}_y \Psi_s(\mathbf{x}, y) \quad [5.38]$$

$$\hat{y}_T = \operatorname{argmax}_y \Psi_t([\mathbf{x}; \hat{y}_S], y). \quad [5.39]$$

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),

$$\ell(\boldsymbol{\theta}_t) = \sum_{i=1}^N \ell^{(i)}(\mathbf{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}_t) + \lambda \|\boldsymbol{\theta}_t - \boldsymbol{\theta}_s\|_2^2, \quad [5.40]$$

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 vector would be,

$$\begin{aligned} \mathbf{f}(\mathbf{x}, y, d) = & \{(boring, \odot, \text{MOVIE}) : 1, (boring, \odot, *) : 1, \\ & (flavorless, \odot, \text{MOVIE}) : 1, (flavorless, \odot, *) : 1, \\ & (three\text{-}day\text{-}old, \odot, \text{MOVIE}) : 1, (three\text{-}day\text{-}old, \odot, *) : 1, \\ & \dots\}, \end{aligned}$$

with $(boring, \odot, \text{MOVIE})$ indicating the word *boring* appearing in a negative labeled document in the MOVIE domain, and $(boring, \odot, *)$ 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,

$$\mathbf{g}(\mathbf{x}^{(i)}) = \mathbf{U}\mathbf{x}^{(i)}. \quad [5.41]$$

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

⁸<http://www.colesmithey.com/capsules/2017/06/wonder-woman.HTML>, accessed October 9, 2017.

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).

2868 include evaluative adjectives like *outstanding* and *disappointing* (Blitzer et al., 2007). For
 2869 each pivot feature j , we define an auxiliary problem of predicting whether the feature is
 2870 present in each example, using the remaining base features. Let ϕ_j denote the weights of
 2871 this classifier, and us horizontally concatenate the weights for each of the N_p pivot features
 2872 into a matrix $\Phi = [\phi_1, \phi_2, \dots, \phi_{N_p}]$.

2873 We then perform truncated singular value decomposition on Φ , as described in § 5.2.1,
 2874 obtaining $\Phi \approx \mathbf{U}\mathbf{S}\mathbf{V}^\top$. The rows of the matrix \mathbf{U} summarize information about each base
 2875 feature: indeed, the truncated singular value decomposition identifies a low-dimension
 2876 basis for the weight matrix Φ , which in turn links base features to pivot features. Sup-
 2877 pose that a base feature *reliable* occurs only in the target domain of appliance reviews.
 2878 Nonetheless, it will have a positive weight towards some pivot features (e.g., *outstanding*,
 2879 *recommended*), and a negative weight towards others (e.g., *worthless*, *unpleasant*). A base
 2880 feature such as *watchable* might have the same associations with the pivot features, and
 2881 therefore, $u_{\text{reliable}} \approx u_{\text{watchable}}$. The matrix \mathbf{U} can thus project the base features into a
 2882 space in which this information is shared.

2883 5.4.2.2 Non-linear projection

2884 Non-linear transformations of the base features can be accomplished by implementing
 2885 the transformation function as a deep neural network, which is trained from an auxiliary
 2886 objective.

2887 **Denoising objectives** One possibility is to train a projection function to reconstruct a
 2888 corrupted version of the original input. The original input can be corrupted in various
 2889 ways: by the addition of random noise (Glorot et al., 2011; Chen et al., 2012), or by the
 2890 deletion of features (Chen et al., 2012; Yang and Eisenstein, 2015). Denoising objectives
 2891 share many properties of the linear projection method described above: they enable the
 2892 projection function to be trained on large amounts of unlabeled data from the target do-
 2893 main, and allow information to be shared across the feature space, thereby reducing sen-
 2894 sitivity to rare and domain-specific features.

2895 **Adversarial objectives** The ultimate goal is for the transformed representations $g(\mathbf{x}^{(i)})$
 2896 to be domain-general. This can be made an explicit optimization criterion by comput-
 2897 ing the similarity of transformed instances both within and between domains (Tzeng
 2898 et al., 2015), or by formulating an auxiliary classification task, in which the domain it-
 2899 self is treated as a label (Ganin et al., 2016). This setting is **adversarial**, because we want
 2900 to learn a representation that makes this classifier perform poorly. At the same time, we
 2901 want $g(\mathbf{x}^{(i)})$ to enable accurate predictions of the labels $y^{(i)}$.

2902 To formalize this idea, let $d^{(i)}$ represent the domain of instance i , and let $\ell_d(g(\mathbf{x}^{(i)}), d^{(i)}; \theta_d)$
 2903 represent the loss of a classifier (typically a deep neural network) trained to predict $d^{(i)}$

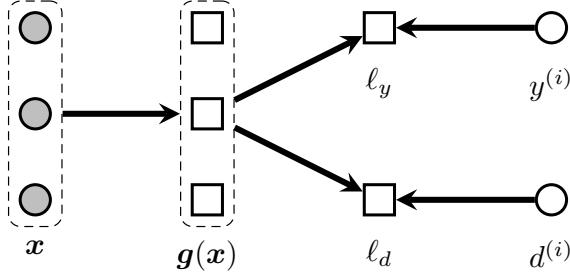


Figure 5.4: A schematic view of adversarial domain adaptation. The loss ℓ_y is computed only for instances from the source domain, where labels $y^{(i)}$ are available.

from the transformed representation $g(x^{(i)})$, using parameters θ_d . Analogously, let $\ell_y(g(x^{(i)}), y^{(i)}; \theta_y)$ represent the loss of a classifier trained to predict the label $y^{(i)}$ from $g(x^{(i)})$, using parameters 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,

$$\min_{f, \theta_g, \theta_y, \theta_d} \sum_{i=1}^{N_\ell + N_u} \ell_d(g(x^{(i)}; \theta_g), d^{(i)}; \theta_d) - \sum_{i=1}^{N_\ell} \ell_y(g(x^{(i)}), y^{(i)}; \theta_y), \quad [5.42]$$

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,

2925 this procedure will eventually converge to drawing samples from the true posterior over
 2926 the missing data, $p(z^{(1:N_z)} | \mathbf{x}^{(1:N_x)})$. For example, in the case of clustering, the missing
 2927 data $z^{(1:N_z)}$ is the set of cluster memberships, $\mathbf{y}^{(1:N)}$, so we draw samples from the pos-
 2928 terior distribution over clusterings of the data. If a single clustering is required, we can
 2929 select the one with the highest conditional likelihood, $\hat{z} = \operatorname{argmax}_z p(z^{(1:N_z)} | \mathbf{x}^{(1:N_x)})$.

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,

$$z^{(n)} | \mathbf{x}, \mathbf{z}^{(-n)} \sim p(z^{(n)} | \mathbf{x}, \mathbf{z}^{(-n)}), \quad [5.43]$$

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, $p(z^{(1:N_z)} | \mathbf{x}^{(1:N_x)})$. In probabilistic clustering, the sampling distribution has the following form,

$$p(z^{(i)} | \mathbf{x}, \mathbf{z}^{(-i)}) = \frac{p(\mathbf{x}^{(i)} | z^{(i)}; \boldsymbol{\phi}) \times p(z^{(i)}; \boldsymbol{\mu})}{\sum_{z=1}^K p(\mathbf{x}^{(i)} | z; \boldsymbol{\phi}) \times p(z; \boldsymbol{\mu})} \quad [5.44]$$

$$\propto \text{Multinomial}(\mathbf{x}^{(i)}; \boldsymbol{\phi}_{z^{(i)}}) \times \boldsymbol{\mu}_{z^{(i)}}. \quad [5.45]$$

2930 In this case, the sampling distribution does not depend on the other instances $\mathbf{x}^{(-i)}, \mathbf{z}^{(-i)}$:
 2931 given the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$, the posterior distribution over each $z^{(i)}$ can be computed
 2932 from $\mathbf{x}^{(i)}$ alone.

2933 In sampling algorithms, there are several choices for how to deal with the parameters.
 2934 One possibility is to sample them too. To do this, we must add them to the generative
 2935 story, by introducing a prior distribution. For the multinomial and categorical parameters
 2936 in the EM clustering model, the **Dirichlet distribution** is a typical choice, since it defines
 2937 a probability on exactly the set of vectors that can be parameters: vectors that sum to one
 2938 and include only non-negative numbers.¹⁰

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,

$$p_{\text{Dirichlet}}(\boldsymbol{\theta} | \boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^K \theta_i^{\alpha_i - 1} \quad [5.46]$$

$$B(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^K \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^K \alpha_i)}, \quad [5.47]$$

with $\Gamma(\cdot)$ indicating the gamma function, a generalization of the factorial function to non-negative reals.

2939 To incorporate this prior, the generative model must augmented to indicate that each
 2940 $\phi_z \sim \text{Dirichlet}(\alpha_\phi)$, and $\mu \sim \text{Dirichlet}(\alpha_\mu)$. The hyperparameters α are typically set to
 2941 a constant vector $\alpha = [\alpha, \alpha, \dots, \alpha]$. When α is large, the Dirichlet distribution tends to
 2942 generate vectors that are nearly uniform; when α is small, it tends to generate vectors that
 2943 assign most of their probability mass to a few entries. Given prior distributions over ϕ
 2944 and μ , we can now include them in Gibbs sampling, drawing values for these parameters
 2945 from posterior distributions that are conditioned on the other variables in the model.

2946 Unfortunately, sampling ϕ and μ usually leads to slow convergence, meaning that a
 2947 large number of samples is required before the Markov chain breaks free from the initial
 2948 conditions. The reason is that the sampling distributions for these parameters are tightly
 2949 constrained by the cluster memberships $y^{(i)}$, which in turn are tightly constrained by the
 2950 parameters. There are two solutions that are frequently employed:

- 2951 • **Empirical Bayesian** methods maintain ϕ and μ as parameters rather than latent
 2952 variables. They still employ sampling in the E-step of the EM algorithm, but they
 2953 update the parameters using expected counts that are computed from the samples
 2954 rather than from parametric distributions. This EM-MCMC hybrid is also known
 2955 as Monte Carlo Expectation Maximization (MCEM; Wei and Tanner, 1990), and is
 2956 well-suited for cases in which it is difficult to compute $q^{(i)}$ directly.
- 2957 • In **collapsed Gibbs sampling**, we analytically integrate ϕ and μ out of the model.
 2958 The cluster memberships $y^{(i)}$ are the only remaining latent variable; we sample them
 2959 from the compound distribution,

$$p(y^{(i)} | \mathbf{x}^{(1:N)}, \mathbf{y}^{(-i)}; \alpha_\phi, \alpha_\mu) = \int_{\phi, \mu} p(\phi, \mu | \mathbf{y}^{(-i)}, \mathbf{x}^{(1:N)}; \alpha_\phi, \alpha_\mu) p(y^{(i)} | \mathbf{x}^{(1:N)}, \mathbf{y}^{(-i)}, \phi, \mu) d\phi d\mu. \quad [5.48]$$

2960 For multinomial and Dirichlet distributions, this integral can be computed in closed
 2961 form.

2962 MCMC algorithms are guaranteed to converge to the true posterior distribution over
 2963 the latent variables, but there is no way to know how long this will take. In practice, the
 2964 rate of convergence depends on initialization, just as expectation-maximization depends
 2965 on initialization to avoid local optima. Thus, while Gibbs Sampling and other MCMC
 2966 algorithms provide a powerful and flexible array of techniques for statistical inference in
 2967 latent variable models, they are not a panacea for the problems experienced by EM.

2968 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,

$$\mathbf{C} = \sum_{i=1}^N \bar{\mathbf{x}}^{(i)} (\bar{\mathbf{x}}^{(i)})^\top. \quad [5.49]$$

The expected value of this matrix under $p(\mathbf{x} | \phi, \mu)$, as

$$E[\mathbf{C}] = \sum_{i=1}^N \sum_{k=1}^K \Pr(Z^{(i)} = k; \boldsymbol{\mu}) \phi_k \phi_k^\top \quad [5.50]$$

$$= \sum_k^K N \mu_k \phi_k \phi_k^\top \quad [5.51]$$

$$= \Phi \text{Diag}(N\mu) \Phi^\top, \quad [5.52]$$

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,

$$\mathbf{C} = \Phi \text{Diag}(N\mu) \Phi^\top, \quad [5.53]$$

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:

$$p(z^{(i)} | \mathbf{x}^{(i)}; \phi, \mu) \propto p(\mathbf{x}^{(i)} | z^{(i)}; \phi) \times p(z^{(i)}; \mu). \quad [5.54]$$

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).

2987 **Additional resources**

2988 There are a number of other learning paradigms that deviate from supervised learning.

- 2989 • **Active learning:** the learner selects unlabeled instances and requests annotations (Set-
- 2990 tles, 2012).
- 2991 • **Multiple instance learning:** labels are applied to bags of instances, with a positive
- 2992 label applied if at least one instance in the bag meets the criterion (Dietterich et al.,
- 2993 1997; Maron and Lozano-Pérez, 1998).
- 2994 • **Constraint-driven learning:** supervision is provided in the form of explicit con-
- 2995 straints on the learner (Chang et al., 2007; Ganchev et al., 2010).
- 2996 • **Distant supervision:** noisy labels are generated from an external resource (Mintz
- 2997 et al., 2009, also see § 17.2.3).
- 2998 • **Multitask learning:** the learner induces a representation that can be used to solve
- 2999 multiple classification tasks (Collobert et al., 2011).
- 3000 • **Transfer learning:** the learner must solve a classification task that differs from the
- 3001 labeled data (Pan and Yang, 2010).

3002 Expectation-maximization was introduced by Dempster et al. (1977), and is discussed

3003 in more detail by Murphy (2012). Like most machine learning treatments, Murphy focus

3004 on continuous observations and Gaussian likelihoods, rather than the discrete observa-

3005 tions typically encountered in natural language processing. Murphy (2012) also includes

3006 an excellent chapter on MCMC; for a textbook-length treatment, see Robert and Casella

3007 (2013). For still more on Bayesian latent variable models, see Barber (2012), and for ap-

3008 plications of Bayesian models to natural language processing, see Cohen (2016). Surveys

3009 are available for semi-supervised learning (Zhu and Goldberg, 2009) and domain adapta-

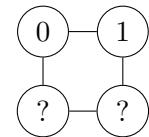
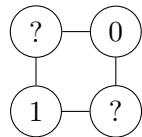
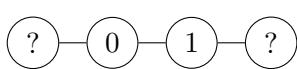
3010 tion (Søgaard, 2013), although both pre-date the current wave of interest in deep learning.

3011 **Exercises**

- 3012 1. Derive the expectation maximization update for the parameter μ in the EM cluster-
- 3013 ing model.
- 3014 2. Derive the E-step and M-step updates for the following generative model. You may
- 3015 assume that the labels $y^{(i)}$ are observed, but $z_m^{(i)}$ is not.
 - 3016 • For each instance i ,
 - 3017 – Draw label $y^{(i)} \sim \text{Categorical}(\boldsymbol{\mu})$
 - 3018 – For each token $m \in \{1, 2, \dots, M^{(i)}\}$

- 3019 * Draw $z_m^{(i)} \sim \text{Categorical}(\pi)$
 3020 * If $z_m^{(i)} = 0$, draw the current token from a label-specific distribution,
 3021 $w_m^{(i)} \sim \phi_{y^{(i)}}$
 3022 * If $z_m^{(i)} = 1$, draw the current token from a document-specific distribu-
 3023 tion, $w_m^{(i)} \sim \nu^{(i)}$

- 3024 3. Using the iterative updates in Equations 5.34-5.36, compute the outcome of the label
 3025 propagation algorithm for the following examples.



3026 The value inside the node indicates the label, $y^{(i)} \in \{0, 1\}$, with $y^{(i)} = ?$ for unlabeled
 3027 nodes. The presence of an edge between two nodes indicates $w_{i,j} = 1$, and the
 3028 absence of an edge indicates $w_{i,j} = 0$. For the third example, you need only compute
 3029 the first three iterations, and then you can guess at the solution in the limit.

- 3030 4. Use expectation-maximization clustering to train a word-sense induction system,
 3031 applied to the word *say*.

- 3032 • Import `nltk`, run `nltk.download()` and select `semcor`. Import `semcor`
 3033 from `nltk.corpus`.
- 3034 • The command `semcor.tagged_sentences(tag='sense')` returns an iter-
 3035 ator over sense-tagged sentences in the corpus. Each sentence can be viewed as
 3036 an iterator over `tree` objects. For `tree` objects that are sense-annotated words,
 3037 you can access the annotation as `tree.label()`, and the word itself with
 3038 `tree.leaves()`. So `semcor.tagged_sentences(tag='sense')[0][2].label()`
 3039 would return the sense annotation of the third word in the first sentence.
- 3040 • Extract all sentences containing the senses `say.v.01` and `say.v.02`.
- 3041 • Build bag-of-words vectors $x^{(i)}$, containing the counts of other words in those
 3042 sentences, including all words that occur in at least two sentences.
- 3043 • Implement and run expectation-maximization clustering on the merged data.
- 3044 • Compute the frequency with which each cluster includes instances of `say.v.01`
 3045 and `say.v.02`.

3046 In the remaining exercises, you will try out some approaches for semisupervised learn-
 3047 ing and domain adaptation. You will need datasets in multiple domains. You can obtain
 3048 product reviews in multiple domains here: <https://www.cs.jhu.edu/~mdredze/>

3049 datasets/sentiment/processed_acl.tar.gz. Choose a source and target domain,
 3050 e.g. dvds and books, and divide the data for the target domain into training and test sets
 3051 of equal size.

- 3052 5. First, quantify the cost of cross-domain transfer.
- 3053 • Train a logistic regression classifier on the source domain training set, and eval-
 - 3054 uate it on the target domain test set.
 - 3055 • Train a logistic regression classifier on the target domain training set, and eval-
 - 3056 uate it on the target domain test set. This is the “direct transfer” baseline.

3057 Compute the difference in accuracy, which is a measure of the transfer loss across
 3058 domains.

- 3059 6. Next, apply the **label propagation** algorithm from § 5.3.2.
- 3060 As a baseline, using only 5% of the target domain training set, train a classifier, and
 3061 compute its accuracy on the target domain test set.

3062 Next, apply label propagation:

- 3063 • Compute the label matrix \mathbf{Q}_L for the labeled data (5% of the target domain
 3064 training set), with each row equal to an indicator vector for the label (positive
 3065 or negative).
- 3066 • Iterate through the target domain instances, including both test and training
 3067 data. At each instance i , compute all w_{ij} , using Equation 5.32, with $\alpha = 0.01$.
 3068 Use these values to fill in column i of the transition matrix \mathbf{T} , setting all but the
 3069 ten largest values to zero for each column i . Be sure to normalize the column
 3070 so that the remaining values sum to one. You may need to use a sparse matrix
 3071 for this to fit into memory.
- 3072 • Apply the iterative updates from Equations 5.34–5.36 to compute the outcome
 3073 of the label propagation algorithm for the unlabeled examples.

3074 Select the test set instances from \mathbf{Q}_U , and compute the accuracy of this method.
 3075 Compare with the supervised classifier trained only on the 5% sample of the target
 3076 domain training set.

- 3077 7. Using only 5% of the target domain training data (and all of the source domain train-
 3078 ing data), implement one of the supervised domain adaptation baselines in § 5.4.1.
 3079 See if this improves on the “direct transfer” baseline from the previous problem
- 3080 8. Implement EasyAdapt (§ 5.4.1), again using 5% of the target domain training data
 3081 and all of the source domain data.

3082 9. Now try unsupervised domain adaptation, using the “linear projection” method
3083 described in § 5.4.2. Specifically:

- 3084 • Identify 500 pivot features as the words with the highest frequency in the (com-
3085 plete) training data for the source and target domains. Specifically, let x_i^d be the
3086 count of the word i in domain d : choose the 500 words with the largest values
3087 of $\min(x_i^{\text{source}}, x_i^{\text{target}})$.
- 3088 • Train a classifier to predict each pivot feature from the remaining words in the
3089 document.
- 3090 • Arrange the features of these classifiers into a matrix Φ , and perform truncated
3091 singular value decomposition, with $k = 20$
- 3092 • Train a classifier from the source domain data, using the combined features
3093 $\mathbf{x}^{(i)} \oplus \mathbf{U}^\top \mathbf{x}^{(i)}$ — these include the original bag-of-words features, plus the pro-
3094 jected features.
- 3095 • Apply this classifier to the target domain test set, and compute the accuracy.

3096

Part II

3097

Sequences and trees

3098

Chapter 6

3099

Language models

3100 In probabilistic classification, the problem is to compute the probability of a label, conditioned
3101 on the text. Let's now consider the inverse problem: computing the probability of
3102 text itself. Specifically, we will consider models that assign probability to a sequence of
3103 word tokens, $p(w_1, w_2, \dots, w_M)$, with $w_m \in \mathcal{V}$. The set \mathcal{V} is a discrete vocabulary,

$$\mathcal{V} = \{aardvark, abacus, \dots, zither\}. \quad [6.1]$$

3104 Why would you want to compute the probability of a word sequence? In many applications,
3105 the goal is to produce word sequences as output:

- 3106 • In **machine translation** (chapter 18), we convert from text in a source language to
3107 text in a target language.
- 3108 • In **speech recognition**, we convert from audio signal to text.
- 3109 • In **summarization** (§ 16.3.4.1; § 19.2), we convert from long texts into short texts.
- 3110 • In **dialogue systems** (§ 19.3), we convert from the user's input (and perhaps an
3111 external knowledge base) into a text response.

3112 In many of the systems for performing these tasks, there is a subcomponent that computes
3113 the probability of the output text. The purpose of this component is to generate
3114 texts that are more **fluent**. For example, suppose we want to translate a sentence from
3115 Spanish to English.

3116 (6.1) El cafe negro me gusta mucho.

3117 Here is a literal word-for-word translation (a **gloss**):

3118 (6.2) The coffee black me pleases much.

3119 A good language model of English will tell us that the probability of this translation is
 3120 low, in comparison with more grammatical alternatives,

$$p(\text{The coffee black me pleases much}) < p(\text{I love dark coffee}). \quad [6.2]$$

3121 How can we use this fact? Warren Weaver, one of the early leaders in machine trans-
 3122 lation, viewed it as a problem of breaking a secret code (Weaver, 1955):

3123 When I look at an article in Russian, I say: 'This is really written in English,
 3124 but it has been coded in some strange symbols. I will now proceed to decode.'

3125 This observation motivates a generative model (like Naïve Bayes):

3126 • The English sentence $w^{(e)}$ is generated from a **language model**, $p_e(w^{(e)})$.

3127 • The Spanish sentence $w^{(s)}$ is then generated from a **translation model**, $p_{s|e}(w^{(s)} | w^{(e)})$.

Given these two distributions, translation can be performed by Bayes' rule:

$$p_{e|s}(w^{(e)} | w^{(s)}) \propto p_{e,s}(w^{(e)}, w^{(s)}) \quad [6.3]$$

$$= p_{s|e}(w^{(s)} | w^{(e)}) \times p_e(w^{(e)}). \quad [6.4]$$

3128 This is sometimes called the **noisy channel model**, because it envisions English text
 3129 turning into Spanish by passing through a noisy channel, $p_{s|e}$. What is the advantage of
 3130 modeling translation this way, as opposed to modeling $p_{e|s}$ directly? The crucial point is
 3131 that the two distributions $p_{s|e}$ (the translation model) and p_e (the language model) can be
 3132 estimated from separate data. The translation model requires examples of correct trans-
 3133 lations, but the language model requires only text in English. Such monolingual data is
 3134 much more widely available. Furthermore, once estimated, the language model p_e can
 3135 be reused in any application that involves generating English text, including translation
 3136 from other languages.

3137 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**. Consider the quote, attributed to Picasso, "*computers are useless, they can only give you answers.*" One way to estimate the probability of this sentence is,

$$\begin{aligned} p(\text{Computers are useless, they can only give you answers}) \\ = \frac{\text{count}(\text{Computers are useless, they can only give you answers})}{\text{count}(\text{all sentences ever spoken})} \end{aligned} \quad [6.5]$$

3138 This estimator is **unbiased**: in the theoretical limit of infinite data, the estimate will
 3139 be correct. But in practice, we are asking for accurate counts over an infinite number of
 3140 events, since sequences of words can be arbitrarily long. Even with an aggressive upper
 3141 bound of, say, $M = 20$ tokens in the sequence, the number of possible sequences is V^{20} ,
 3142 where $V = |\mathcal{V}|$. A small vocabulary for English would have $V = 10^5$, so there are 10^{100}
 3143 possible sequences. Clearly, this estimator is very data-hungry, and suffers from high vari-
 3144 ance: even grammatical sentences will have probability zero if they have not occurred in
 3145 the training data.¹ We therefore need to introduce bias to have a chance of making reli-
 3146 able estimates from finite training data. The language models that follow in this chapter
 3147 introduce bias in various 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 $p(\mathbf{w}) = p(w_1, w_2, \dots, w_M)$ can be refactored using the chain rule (see § A.2):

$$p(\mathbf{w}) = p(w_1, w_2, \dots, w_M) \quad [6.6]$$

$$= p(w_1) \times p(w_2 | w_1) \times p(w_3 | w_2, w_1) \times \dots \times p(w_M | w_{M-1}, \dots, w_1) \quad [6.7]$$

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,

$$\begin{aligned} p(\text{useless} | \text{computers are}) &= \frac{\text{count}(\text{computers are useless})}{\sum_{x \in \mathcal{V}} \text{count}(\text{computers are } x)} \\ &= \frac{\text{count}(\text{computers are useless})}{\text{count}(\text{computers are})}. \end{aligned}$$

3148 We haven't made any approximations yet, and we could have just as well applied the
 3149 chain rule in reverse order,

$$p(\mathbf{w}) = p(w_M) \times p(w_{M-1} | w_M) \times \dots \times p(w_1 | w_2, \dots, w_M), \quad [6.8]$$

3150 or in any other order. But this means that we also haven't really made any progress:
 3151 to compute the conditional probability $p(w_M | w_{M-1}, w_{M-2}, \dots, w_1)$, we would need to
 3152 model V^{M-1} contexts. Such a distribution cannot be estimated from any realistic sample
 3153 of text.

To solve this problem, n -gram models make a crucial simplifying approximation: they condition on only the past $n - 1$ words.

$$p(w_m | w_{m-1} \dots w_1) \approx p(w_m | w_{m-1}, \dots, w_{m-n+1}) \quad [6.9]$$

¹Chomsky 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*.

This means that the probability of a sentence w can be approximated as

$$p(w_1, \dots, w_M) \approx \prod_{m=1}^M p(w_m | w_{m-1}, \dots, w_{m-n+1}) \quad [6.10]$$

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:

$$p(I \text{ like black coffee}) = p(I | \square) \times p(\text{like} | I) \times p(\text{black} | \text{like}) \times p(\text{coffee} | \text{black}) \times p(\blacksquare | \text{coffee}). \quad [6.11]$$

3154 This model requires estimating and storing the probability of only V^n events, which is
 3155 exponential in the order of the n -gram, and not V^M , which is exponential in the length of
 3156 the sentence. The n -gram probabilities can be computed by relative frequency estimation,

$$p(w_m | w_{m-1}, w_{m-2}) = \frac{\text{count}(w_{m-2}, w_{m-1}, w_m)}{\sum_{w'} \text{count}(w_{m-2}, w_{m-1}, w')} \quad [6.12]$$

3157 The hyperparameter n controls the size of the context used in each conditional proba-
 3158 bility. If this is misspecified, the language model will perform poorly. Let's consider the
 3159 potential problems concretely.

3160 **When n is too small.** Consider the following sentences:

3161 (6.3) **Gorillas** always like to groom **their** friends.

3162 (6.4) The **computer** that's on the 3rd floor of our office building **crashed**.

3163 In each example, the words written in bold depend on each other: the likelihood
 3164 of *their* depends on knowing that *gorillas* is plural, and the likelihood of *crashed* de-
 3165 pends on knowing that the subject is a *computer*. If the n -grams are not big enough
 3166 to capture this context, then the resulting language model would offer probabili-
 3167 ties that are too low for these sentences, and too high for sentences that fail basic
 3168 linguistic tests like number agreement.

3169 **When n is too big.** In this case, it is hard to get good estimates of the n -gram parameters from
 3170 our dataset, because of data sparsity. To handle the *gorilla* example, it is necessary to
 3171 model 6-grams, which means accounting for V^6 events. Under a very small vocab-
 3172 uary of $V = 10^4$, this means estimating the probability of 10^{24} distinct events.

3173 These two problems point to another **bias-variance tradeoff** (see § 2.1.4). A small n -
 3174 gram size introduces high bias, and a large n -gram size introduces high variance. We
 3175 can even have both problems at the same time! Language is full of long-range dependen-
 3176 cies that we cannot capture because n is too small; at the same time, language datasets
 3177 are full of rare phenomena, whose probabilities we fail to estimate accurately because n
 3178 is too large. One solution is to try to keep n large, while still making low-variance esti-
 3179 mates of the underlying parameters. To do this, we will introduce a different sort of bias:
 3180 **smoothing**.

3181 6.2 Smoothing and discounting

3182 Limited data is a persistent problem in estimating language models. In § 6.1, we pre-
 3183 sented n -grams as a partial solution. Bit sparse data can be a problem even for low-order
 3184 n -grams; at the same time, many linguistic phenomena, like subject-verb agreement, can-
 3185 not be incorporated into language models without high-order n -grams. It is therefore
 3186 necessary to add additional inductive biases to n -gram language models. This section
 3187 covers some of the most intuitive and common approaches, but there are many more (see
 3188 Chen and Goodman, 1999).

3189 6.2.1 Smoothing

3190 A major concern in language modeling is to avoid the situation $p(w) = 0$, which could
 3191 arise as a result of a single unseen n-gram. A similar problem arose in Naïve Bayes, and
 3192 the solution was **smoothing**: adding imaginary “pseudo” counts. The same idea can be
 3193 applied to n -gram language models, as shown here in the bigram case,

$$p_{\text{smooth}}(w_m \mid w_{m-1}) = \frac{\text{count}(w_{m-1}, w_m) + \alpha}{\sum_{w' \in \mathcal{V}} \text{count}(w_{m-1}, w') + V\alpha}. \quad [6.13]$$

3194 This basic framework is called **Lidstone smoothing**, but special cases have other names:

- 3195 • **Laplace smoothing** corresponds to the case $\alpha = 1$.
- 3196 • **Jeffreys-Perks law** corresponds to the case $\alpha = 0.5$, which works well in practice
 3197 and benefits from some theoretical justification (Manning and Schütze, 1999).

3198 To ensure that the probabilities are properly normalized, anything that we add to the
 3199 numerator (α) must also appear in the denominator ($V\alpha$). This idea is reflected in the
 3200 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,

$$d_i = \frac{c_i^*}{c_i} = \frac{(c_i + \alpha)}{c_i} \frac{M}{(M + V\alpha)}.$$

3201 6.2.2 Discounting and backoff

3202 Discounting “borrows” probability mass from observed n -grams and redistributes it. In
 3203 Lidstone smoothing, the borrowing is done by increasing the denominator of the relative
 3204 frequency estimates. The borrowed probability mass is then redistributed by increasing
 3205 the numerator for all n -grams. Another approach would be to borrow the same amount
 3206 of probability mass from all observed n -grams, and redistribute it among only the unob-
 3207 served n -grams. This is called **absolute discounting**. For example, suppose we set an
 3208 absolute discount $d = 0.1$ in a bigram model, and then redistribute this probability mass
 3209 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,

$$c^*(i, j) = c(i, j) - d \quad [6.15]$$

$$p_{\text{Katz}}(i | j) = \begin{cases} \frac{c^*(i, j)}{c(j)} & \text{if } c(i, j) > 0 \\ \alpha(j) \times \frac{p_{\text{unigram}}(i)}{\sum_{i': c(i', j)=0} p_{\text{unigram}}(i')} & \text{if } c(i, j) = 0. \end{cases} \quad [6.16]$$

3210 The term $\alpha(j)$ indicates the amount of probability mass that has been discounted for
 3211 context j . This probability mass is then divided across all the unseen events, $\{i' : c(i', j) =$
 3212 $0\}$, proportional to the unigram probability of each word i' . The discount parameter d can
 3213 be optimized to maximize performance (typically held-out log-likelihood) on a develop-
 3214 ment set.

3215 6.2.3 *Interpolation

3216 Backoff is one way to combine different order n -gram models. An alternative approach
 3217 is **interpolation**: setting the probability of a word in context to a weighted sum of its
 3218 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,

$$\begin{aligned} p_{\text{Interpolation}}(w_m | w_{m-1}, w_{m-2}) &= \lambda_3 p_3^*(w_m | w_{m-1}, w_{m-2}) \\ &\quad + \lambda_2 p_2^*(w_m | w_{m-1}) \\ &\quad + \lambda_1 p_1^*(w_m). \end{aligned}$$

3219 In this equation, p_n^* is the unsmoothed empirical probability given by an n -gram lan-
 3220 guage model, and λ_n is the weight assigned to this model. To ensure that the interpolated
 3221 $p(w)$ is still a valid probability distribution, the values of λ must obey the constraint,
 3222 $\sum_{n=1}^{n_{\max}} \lambda_n = 1$. But how to find the specific values?

3223 An elegant solution is **expectation-maximization**. Recall from chapter 5 that we can
 3224 think about EM as learning with *missing data*: we just need to choose missing data such
 3225 that learning would be easy if it weren't missing. What's missing in this case? Think of
 3226 each word w_m as drawn from an n -gram of unknown size, $z_m \in \{1 \dots n_{\max}\}$. This z_m is
 3227 the missing data that we are looking for. Therefore, the application of EM to this problem
 3228 involves the following **generative process**:

3229 **for** Each token $w_m, m = 1, 2, \dots, M$ **do**:
 3230 draw the n -gram size $z_m \sim \text{Categorical}(\lambda)$;
 3231 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 ,

$$\lambda_z \propto \sum_{m=1}^M q_m(z). \quad [6.22]$$

3233 A solution is obtained by iterating between updates to q and λ . The complete algorithm
 3234 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  $\lambda$ 
  
```

3235 **6.2.4 *Kneser-Ney smoothing**

3236 Kneser-Ney smoothing is based on absolute discounting, but it redistributes the result-
 3237 ing probability mass in a different way from Katz backoff. Empirical evidence points
 3238 to Kneser-Ney smoothing as the state-of-art for n -gram language modeling (Goodman,
 3239 2001). To motivate Kneser-Ney smoothing, consider the example: *I recently visited ..*
 3240 Which of the following is more likely?

- 3241 • *Francisco*
 3242 • *Duluth*

3243 Now suppose that both bigrams *visited Duluth* and *visited Francisco* are unobserved in
 3244 the training data, and furthermore, the unigram probability $p_1^*(\text{Francisco})$ is greater than
 3245 $p^*(\text{Duluth})$. Nonetheless we would still guess that $p(\text{visited Duluth}) > p(\text{visited Francisco})$,
 3246 because *Duluth* is a more “versatile” word: it can occur in many contexts, while *Francisco*
 3247 usually occurs in a single context, following the word *San*. This notion of versatility is the
 3248 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

$$p_{KN}(w | u) = \begin{cases} \frac{\text{count}(w, u) - d}{\text{count}(u)}, & \text{count}(w, u) > 0 \\ \alpha(u) \times p_{\text{continuation}}(w), & \text{otherwise} \end{cases} \quad [6.23]$$

$$p_{\text{continuation}}(w) = \frac{|u : \text{count}(w, u) > 0|}{\sum_{w' \in \mathcal{V}} |u' : \text{count}(w', u') > 0|}. \quad [6.24]$$

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,

$$\alpha(u) = |w : \text{count}(w, u) > 0| \times \frac{d}{\text{count}(u)}. \quad [6.25]$$

3249 This is the amount of probability mass left to account for versatility, which we define via
 3250 the *continuation probability* $p_{\text{continuation}}(w)$ as proportional to the number of observed con-
 3251 texts in which w appears. The numerator of the continuation probability is the number of
 3252 contexts u in which w appears; the denominator normalizes the probability by summing
 3253 the same quantity over all words w' .

3254 The idea of modeling versatility by counting contexts may seem heuristic, but there is
 3255 an elegant theoretical justification from Bayesian nonparametrics (Teh, 2006). Kneser-Ney
 3256 smoothing on n -grams was the dominant language modeling technique before the arrival
 3257 of neural language models.

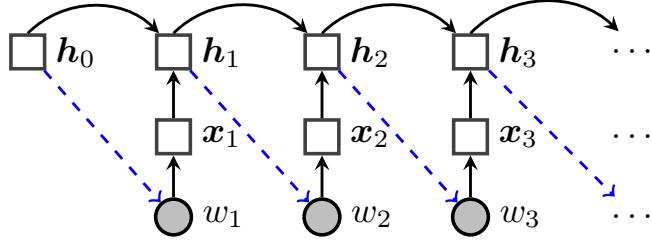


Figure 6.1: The recurrent neural network language model, viewed as an “unrolled” computation graph. Solid lines indicate direct computation, dotted blue lines indicate probabilistic dependencies, circles indicate random variables, and squares indicate computation nodes.

3258 6.3 Recurrent neural network language models

3259 N -gram language models have been largely supplanted by neural networks. These mod-
 3260 els do not make the n -gram assumption of restricted context; indeed, they can incorporate
 3261 arbitrarily distant contextual information, while remaining computationally and statisti-
 3262 cally tractable.

3263 The first insight behind neural language models is to treat word prediction as a *dis-
 3264 criminative* learning task.² The goal is to compute the probability $p(w | u)$, where $w \in \mathcal{V}$ is
 3265 a word, and u is the context, which depends on the previous words. Rather than directly
 3266 estimating the word probabilities from (smoothed) relative frequencies, we can treat
 3267 language modeling as a machine learning problem, and estimate parameters that maxi-
 3268 mize the log conditional probability of a corpus.

3269 The second insight is to reparametrize the probability distribution $p(w | u)$ as a func-
 3270 tion of two dense K -dimensional numerical vectors, $\beta_w \in \mathbb{R}^K$, and $v_u \in \mathbb{R}^K$,

$$p(w | u) = \frac{\exp(\beta_w \cdot v_u)}{\sum_{w' \in \mathcal{V}} \exp(\beta_{w'} \cdot v_u)}, \quad [6.26]$$

3271 where $\beta_w \cdot v_u$ represents a dot product. As usual, the denominator ensures that the prob-
 3272 ability distribution is properly normalized. This vector of probabilities is equivalent to
 3273 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).

3300 **6.3.1 Backpropagation through time**

3301 The recurrent neural network language model has the following parameters:

- 3302 • $\phi_i \in \mathbb{R}^K$, the “input” word vectors (these are sometimes called **word embeddings**,
 3303 since each word is embedded in a K -dimensional space);
 3304 • $\beta_i \in \mathbb{R}^K$, the “output” word vectors;
 3305 • $\Theta \in \mathbb{R}^{K \times K}$, the recurrence operator;
 3306 • \mathbf{h}_0 , the initial state.

3307 Each of these parameters can be estimated by formulating an objective function over the
 3308 training corpus, $L(\mathbf{w})$, and then applying backpropagation to obtain gradients on the
 3309 parameters from a minibatch of training examples (see § 3.3.1). Gradient-based updates
 3310 can be computed from an online learning algorithm such as stochastic gradient descent
 3311 (see § 2.5.2).

3312 The application of backpropagation to recurrent neural networks is known as **back-**
 3313 **propagation through time**, because the gradients on units at time m depend in turn on the
 3314 gradients of units at earlier times $n < m$. Let ℓ_{m+1} represent the negative log-likelihood
 3315 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 :

$$\mathbf{h}_m = g(\mathbf{x}_m, \mathbf{h}_{m-1}) \quad [6.34]$$

$$\frac{\partial h_{m,k}}{\partial \theta_{k,k'}} = g'(\mathbf{x}_{m,k} + \boldsymbol{\theta}_k \cdot \mathbf{h}_{m-1})(h_{m-1,k'} + \boldsymbol{\theta}_k \cdot \frac{\partial \mathbf{h}_{m-1}}{\partial \theta_{k,k'}}), \quad [6.35]$$

3316 where g' is the local derivative of the nonlinear function g . The key point in this equation
 3317 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
 3318 so on, until reaching the initial state \mathbf{h}_0 .

3319 Each derivative $\frac{\partial \mathbf{h}_m}{\partial \theta_{k,k'}}$ will be reused many times: it appears in backpropagation from
 3320 the loss ℓ_m , but also in all subsequent losses $\ell_{n>m}$. Neural network toolkits such as
 3321 Torch (Collobert et al., 2011) and DyNet (Neubig et al., 2017) compute the necessary

derivatives automatically, and cache them for future use. An important distinction from the feedforward neural networks considered in chapter 3 is that the size of the computation graph is not fixed, but varies with the length of the input. This poses difficulties for toolkits that are designed around static computation graphs, such as TensorFlow (Abadi et al., 2016).⁴

6.3.2 Hyperparameters

The RNN language model has several hyperparameters that must be tuned to ensure good performance. The model capacity is controlled by the size of the word and context vectors K , which play a role that is somewhat analogous to the size of the n -gram context. For datasets that are large with respect to the vocabulary (i.e., there is a large token-to-type ratio), we can afford to estimate a model with a large K , which enables more subtle distinctions between words and contexts. When the dataset is relatively small, then K must be smaller too, or else the model may “memorize” the training data, and fail to generalize. Unfortunately, this general advice has not yet been formalized into any concrete formula for choosing K , and trial-and-error is still necessary. Overfitting can also be prevented by **dropout**, which involves randomly setting some elements of the computation to zero (Srivastava et al., 2014), forcing the learner not to rely too much on any particular dimension of the word or context vectors. The dropout rate must also be tuned on development data.

6.3.3 Gated recurrent neural networks

In principle, recurrent neural networks can propagate information across infinitely long sequences. But in practice, repeated applications of the nonlinear recurrence function causes this information to be quickly attenuated. The same problem affects learning: back-propagation can lead to **vanishing gradients** that decay to zero, or **exploding gradients** that increase towards infinity (Bengio et al., 1994). The exploding gradient problem can be addressed by clipping gradients at some maximum value (Pascanu et al., 2013). The other issues must be addressed by altering the model itself.

The **long short-term memory** (LSTM; Hochreiter and Schmidhuber, 1997) is a popular variant of RNNs that is more robust to these problems. This model augments the hidden state \mathbf{h}_m with a **memory cell** c_m . The value of the memory cell at each time m is a gated sum of two quantities: its previous value c_{m-1} , and an “update” \tilde{c}_m , which is computed from the current input x_m and the previous hidden state \mathbf{h}_{m-1} . The next state \mathbf{h}_m is then computed from the memory cell. Because the memory cell is not passed through a non-linear squashing function during the update, it is possible for information to propagate through the network over long distances.

⁴See <https://www.tensorflow.org/tutorials/recurrent> (retrieved Feb 8, 2018).

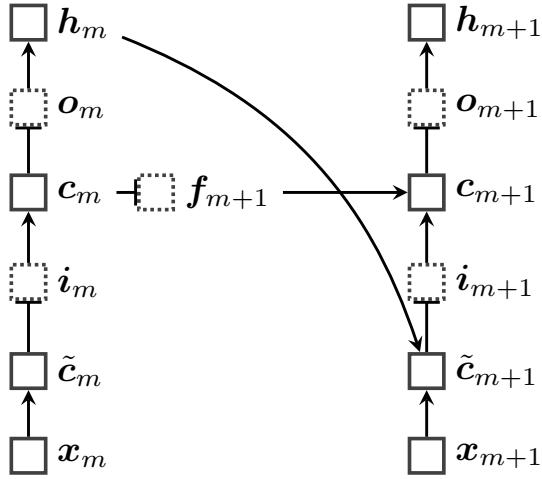


Figure 6.2: The long short-term memory (LSTM) architecture. Gates are shown in boxes with dotted edges. In an LSTM language model, each h_m would be used to predict the next word w_{m+1} .

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:

$$f_{m+1} = \sigma(\Theta^{(h \rightarrow f)} h_m + \Theta^{(x \rightarrow f)} x_{m+1} + b_f) \quad \text{forget gate} \quad [6.36]$$

$$i_{m+1} = \sigma(\Theta^{(h \rightarrow i)} h_m + \Theta^{(x \rightarrow i)} x_{m+1} + b_i) \quad \text{input gate} \quad [6.37]$$

$$\tilde{c}_{m+1} = \tanh(\Theta^{(h \rightarrow c)} h_m + \Theta^{(x \rightarrow c)} x_{m+1}) \quad \text{update candidate} \quad [6.38]$$

$$c_{m+1} = f_{m+1} \odot c_m + i_{m+1} \odot \tilde{c}_{m+1} \quad \text{memory cell update} \quad [6.39]$$

$$o_{m+1} = \sigma(\Theta^{(h \rightarrow o)} h_m + \Theta^{(x \rightarrow o)} x_{m+1} + b_o) \quad \text{output gate} \quad [6.40]$$

$$h_{m+1} = o_{m+1} \odot \tanh(c_{m+1}) \quad \text{output.} \quad [6.41]$$

3356 The operator \odot is an elementwise (Hadamard) product. Each gate is controlled by a vec-
 3357 tor of weights, which parametrize the previous hidden state (e.g., $\Theta^{(h \rightarrow f)}$) and the current
 3358 input (e.g., $\Theta^{(x \rightarrow f)}$), plus a vector offset (e.g., b_f). The overall operation can be infor-
 3359 mally summarized as $(h_m, c_m) = \text{LSTM}(x_m, (h_{m-1}, c_{m-1}))$, with (h_m, c_m) representing
 3360 the LSTM state after reading token m .

3361 The LSTM outperforms standard recurrent neural networks across a wide range of
 3362 problems. It was first used for language modeling by Sundermeyer et al. (2012), but can
 3363 be applied more generally: the vector h_m can be treated as a complete representation of

3364 the input sequence up to position m , and can be used for any labeling task on a sequence
 3365 of tokens, as we will see in the next chapter.

3366 There are several LSTM variants, of which the Gated Recurrent Unit (Cho et al., 2014)
 3367 is one of the more well known. Many software packages implement a variety of RNN
 3368 architectures, so choosing between them is simple from a user’s perspective. Jozefowicz
 3369 et al. (2015) provide an empirical comparison of various modeling choices circa 2015.

3370 6.4 Evaluating language models

3371 Language modeling is not usually an application in itself: language models are typically
 3372 components of larger systems, and they would ideally be evaluated **extrinsically**. This
 3373 means evaluating whether the language model improves performance on the application
 3374 task, such as machine translation or speech recognition. But this is often hard to do, and
 3375 depends on details of the overall system which may be irrelevant to language modeling.
 3376 In contrast, **intrinsic evaluation** is task-neutral. Better performance on intrinsic metrics
 3377 may be expected to improve extrinsic metrics across a variety of tasks, but there is always
 3378 the risk of over-optimizing the intrinsic metric. This section discusses some intrinsic met-
 3379 rics, but keep in mind the importance of performing extrinsic evaluations to ensure that
 3380 intrinsic performance gains carry over to real applications.

3381 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,

$$\ell(\mathbf{w}) = \sum_{m=1}^M \log p(w_m | w_{m-1}, \dots, w_1), \quad [6.42]$$

3382 treating the entire held-out corpus as a single stream of tokens.

3383 Typically, unknown words are mapped to the $\langle \text{UNK} \rangle$ token. This means that we have
 3384 to estimate some probability for $\langle \text{UNK} \rangle$ on the training data. One way to do this is to fix
 3385 the vocabulary \mathcal{V} to the $V - 1$ words with the highest counts in the training data, and then
 3386 convert all other tokens to $\langle \text{UNK} \rangle$. Other strategies for dealing with out-of-vocabulary
 3387 terms are discussed in § 6.5.

3388 **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,

$$\text{Perplex}(\mathbf{w}) = 2^{-\frac{\ell(\mathbf{w})}{M}}, \quad [6.43]$$

3389 where M is the total number of tokens in the held-out corpus.

3390 Lower perplexities correspond to higher likelihoods, so lower scores are better on this
3391 metric — it is better to be less perplexed. Here are some special cases:

- 3392 • In the limit of a perfect language model, probability 1 is assigned to the held-out
3393 corpus, with $\text{Perplex}(\mathbf{w}) = 2^{-\frac{1}{M} \log_2 1} = 2^0 = 1$.
- 3394 • In the opposite limit, probability zero is assigned to the held-out corpus, which cor-
3395 responds to an infinite perplexity, $\text{Perplex}(\mathbf{w}) = 2^{-\frac{1}{M} \log_2 0} = 2^\infty = \infty$.
- Assume a uniform, unigram model in which $p(w_i) = \frac{1}{V}$ for all words in the vocab-
ular. Then,

$$\begin{aligned} \log_2(\mathbf{w}) &= \sum_{m=1}^M \log_2 \frac{1}{V} = - \sum_{m=1}^M \log_2 V = -M \log_2 V \\ \text{Perplex}(\mathbf{w}) &= 2^{\frac{1}{M} M \log_2 V} \\ &= 2^{\log_2 V} \\ &= V. \end{aligned}$$

3396 This is the “worst reasonable case” scenario, since you could build such a language
3397 model without even looking at the data.

3398 In practice, language models tend to give perplexities in the range between 1 and V .
3399 A small benchmark dataset is the **Penn Treebank**, which contains roughly a million to-
3400 kens; its vocabulary is limited to 10,000 words, with all other tokens mapped a special
3401 $\langle \text{UNK} \rangle$ symbol. On this dataset, a well-smoothed 5-gram model achieves a perplexity of
3402 141 (Mikolov and Zweig, Mikolov and Zweig), and an LSTM language model achieves
3403 perplexity of roughly 80 (Zaremba, Sutskever, and Vinyals, Zaremba et al.). Various en-
3404 hancements to the LSTM architecture can bring the perplexity below 60 (Merity et al.,
3405 2018). A larger-scale language modeling dataset is the 1B Word Benchmark (Chelba et al.,
3406 2013), which contains text from Wikipedia. On this dataset, a perplexities of around 25
3407 can be obtained by averaging together multiple LSTM language models (Jozefowicz et al.,
3408 2016).

3409 **6.5 Out-of-vocabulary words**

3410 So far, we have assumed a **closed-vocabulary** setting — the vocabulary \mathcal{V} is assumed to be
 3411 a finite set. In realistic application scenarios, this assumption may not hold. Consider, for
 3412 example, the problem of translating newspaper articles. The following sentence appeared
 3413 in a Reuters article on January 6, 2017:⁵

3414 The report said U.S. intelligence agencies believe Russian military intelligence,
 3415 the **GRU**, used intermediaries such as **WikiLeaks**, **DCLeaks.com** and the **Guc-**
 3416 **cifer** 2.0 "persona" to release emails...

3417 Suppose that you trained a language model on the Gigaword corpus,⁶ which was released
 3418 in 2003. The bolded terms either did not exist at this date, or were not widely known; they
 3419 are unlikely to be in the vocabulary. The same problem can occur for a variety of other
 3420 terms: new technologies, previously unknown individuals, new words (e.g., *hashtag*), and
 3421 numbers.

3422 One solution is to simply mark all such terms with a special token, $\langle \text{UNK} \rangle$. While
 3423 training the language model, we decide in advance on the vocabulary (often the K most
 3424 common terms), and mark all other terms in the training data as $\langle \text{UNK} \rangle$. If we do not want
 3425 to determine the vocabulary size in advance, an alternative approach is to simply mark
 3426 the first occurrence of each word type as $\langle \text{UNK} \rangle$.

3427 But it often better to make distinctions about the likelihood of various unknown words.
 3428 This is particularly important in languages that have rich morphological systems, with
 3429 many inflections for each word. For example, Portuguese is only moderately complex
 3430 from a morphological perspective, yet each verb has dozens of inflected forms (see Fig-
 3431 ure 4.3b). In such languages, there will be many word types that we do not encounter in a
 3432 corpus, which are nonetheless predictable from the morphological rules of the language.
 3433 To use a somewhat contrived English example, if *transfenestrate* is in the vocabulary, our
 3434 language model should assign a non-zero probability to the past tense *transfenestrated*,
 3435 even if it does not appear in the training data.

3436 One way to accomplish this is to supplement word-level language models with **character-**
 3437 **level language models**. Such models can use n -grams or RNNs, but with a fixed vocab-
 3438 uary equal to the set of ASCII or Unicode characters. For example, Ling et al. (2015)
 3439 propose an LSTM model over characters, and Kim (2014) employ a convolutional neural
 3440 network. A more linguistically motivated approach is to segment words into meaningful
 3441 subword units, known as **morphemes** (see chapter 9). For example, Botha and Blunsom

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>

3442 (2014) induce vector representations for morphemes, which they build into a log-bilinear
 3443 language model; Bhatia et al. (2016) incorporate morpheme vectors into an LSTM.

3444 Additional resources

3445 A variety of neural network architectures have been applied to language modeling. No-
 3446 table earlier non-recurrent architectures include the neural probabilistic language model (Ben-
 3447 gio et al., 2003) and the log-bilinear language model (Mnih and Hinton, 2007). Much more
 3448 detail on these models can be found in the text by Goodfellow et al. (2016).

3449 Exercises

- 3450 1. Prove that n -gram language models give valid probabilities if the n -gram probabili-
 3451 ties are valid. Specifically, assume that,

$$\sum_{w_m}^{\mathcal{V}} p(w_m | w_{m-1}, w_{m-2}, \dots, w_{m-n+1}) = 1 \quad [6.44]$$

3452 for all contexts $(w_{m-1}, w_{m-2}, \dots, w_{m-n+1})$. Prove that $\sum_w p_n(w) = 1$ for all $w \in \mathcal{V}^*$,
 3453 where p_n is the probability under an n -gram language model. Your proof should
 3454 proceed by induction. You should handle the start-of-string case $p(w_1 | \underbrace{\square, \dots, \square}_{n-1})$,

3455 but you need not handle the end-of-string token.

- 3456 2. First, show that RNN language models are valid using a similar proof technique to
 3457 the one in the previous problem.

3458 Next, let $p_r(w)$ indicate the probability of w under RNN r . An ensemble of RNN
 3459 language models computes the probability,

$$p(w) = \frac{1}{R} \sum_{r=1}^R p_r(w). \quad [6.45]$$

3460 Does an ensemble of RNN language models compute a valid probability?

- 3461 3. Consider a unigram language model over a vocabulary of size V . Suppose that a
 3462 word appears m times in a corpus with M tokens in total. With Lidstone smoothing
 3463 of α , for what values of m is the smoothed probability greater than the unsmoothed
 3464 probability?
- 3465 4. Consider a simple language in which each token is drawn from the vocabulary \mathcal{V}
 3466 with probability $\frac{1}{V}$, independent of all other tokens.

- Given a corpus of size M , what is the expectation of the fraction of all possible bigrams that have zero count? You may assume V is large enough that $\frac{1}{V} \approx \frac{1}{V-1}$.
5. Continuing the previous problem, determine the value of M such that the fraction of bigrams with zero count is at most $\epsilon \in (0, 1)$. As a hint, you may use the approximation $\ln(1 + \alpha) \approx \alpha$ for $\alpha \approx 0$.
 6. In real languages, word probabilities are neither uniform nor independent. Assume that word probabilities are independent but not uniform, so that in general $p(w) \neq \frac{1}{V}$. Prove that the expected fraction of unseen bigrams will be higher than in the IID case.
 7. Consider a recurrent neural network with a single hidden unit and a sigmoid activation, $h_m = \sigma(\theta h_{m-1} + x_m)$. Prove that if $|\theta| < 1$, then the gradient $\frac{\partial h_m}{\partial h_{m-k}}$ goes to zero as $k \rightarrow \infty$.⁷
 8. **Zipf's law** states that if the word types in a corpus are sorted by frequency, then the frequency of the word at rank r is proportional to r^{-s} , where s is a free parameter, usually around 1. (Another way to view Zipf's law is that a plot of log frequency against log rank will be linear.) Solve for s using the counts of the first and second most frequent words, c_1 and c_2 .
 9. Download the wikitext-2 dataset.⁸ Read in the training data and compute word counts. Estimate the Zipf's law coefficient by,

$$\hat{s} = \exp \left(\frac{(\log r) \cdot (\log c)}{\|\log r\|_2^2} \right), \quad [6.46]$$

where $r = [1, 2, 3, \dots]$ is the vector of ranks of all words in the corpus, and $c = [c_1, c_2, c_3, \dots]$ is the vector of counts of all words in the corpus, sorted in descending order.

Make a log-log plot of the observed counts, and the expected counts according to Zipf's law. The sum $\sum_{r=1}^{\infty} r^s = \zeta(s)$ is the Riemann zeta function, available in python's `scipy` library as `scipy.special.zeta`.

10. Using the Pytorch library, train an LSTM language model from the Wikitext training corpus. After each epoch of training, compute its perplexity on the Wikitext validation corpus. Stop training when the perplexity stops improving.

⁷This proof generalizes to vector hidden units by considering the largest eigenvector of the matrix Θ (Pascanu et al., 2013).

⁸Available at https://github.com/pytorch/examples/tree/master/word_language_model/data/wikitext-2 in September 2018. The dataset is already tokenized, and already replaces rare words with `<UNK>`, so no preprocessing is necessary.

3495 Chapter 7

3496 Sequence labeling

3497 The goal of sequence labeling is to assign tags to words, or more generally, to assign
3498 discrete labels to discrete elements in a sequence. There are many applications of se-
3499 quence labeling in natural language processing, and chapter 8 presents an overview. For
3500 now, we'll focus on the classic problem of **part-of-speech tagging**, which requires tagging
3501 each word by its grammatical category. Coarse-grained grammatical categories include
3502 **NOUNs**, which describe things, properties, or ideas, and **VERBS**, which describe actions
3503 and events. Consider a simple input:

3504 (7.1) They can fish.

3505 A dictionary of coarse-grained part-of-speech tags might include **NOUN** as the only valid
3506 tag for *they*, but both **NOUN** and **VERB** as potential tags for *can* and *fish*. An accurate se-
3507 quence labeling algorithm should select the verb tag for both *can* and *fish* in (7.1), but it
3508 should select noun for the same two words in the phrase *can of fish*.

3509 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:

$$f((\mathbf{w} = \text{they can fish}, m = 1), \text{N}) = (\text{they}, \text{N}) \quad [7.1]$$

$$f((\mathbf{w} = \text{they can fish}, m = 2), \text{V}) = (\text{can}, \text{V}) \quad [7.2]$$

$$f((\mathbf{w} = \text{they can fish}, m = 3), \text{V}) = (\text{fish}, \text{V}). \quad [7.3]$$

3510 Here the feature function takes three arguments as input: the sentence to be tagged (e.g.,
3511 *they can fish*), the proposed tag (e.g., N or V), and the index of the token to which this tag

is applied. This simple feature function then returns a single feature: a tuple including the word to be tagged and the tag that has been proposed. If the vocabulary size is V and the number of tags is K , then there are $V \times K$ features. Each of these features must be assigned a weight. These weights can be learned from a labeled dataset using a classification algorithm such as perceptron, but this isn't necessary in this case: it would be equivalent to define the classification weights directly, with $\theta_{w,y} = 1$ for the tag y most 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:

$$\begin{aligned} f((\mathbf{w} = \text{they can fish}, 1), \mathbf{N}) = & \{(w_m = \text{they}, y_m = \mathbf{N}), \\ & (w_{m-1} = \square, y_m = \mathbf{N}), \\ & (w_{m+1} = \text{can}, y_m = \mathbf{N})\} \end{aligned} \quad [7.4]$$

$$\begin{aligned} f((\mathbf{w} = \text{they can fish}, 2), \mathbf{V}) = & \{(w_m = \text{can}, y_m = \mathbf{V}), \\ & (w_{m-1} = \text{they}, y_m = \mathbf{V}), \\ & (w_{m+1} = \text{fish}, y_m = \mathbf{V})\} \end{aligned} \quad [7.5]$$

$$\begin{aligned} f((\mathbf{w} = \text{they can fish}, 3), \mathbf{V}) = & \{(w_m = \text{fish}, y_m = \mathbf{V}), \\ & (w_{m-1} = \text{can}, y_m = \mathbf{V}), \\ & (w_{m+1} = \blacksquare, y_m = \mathbf{V})\}. \end{aligned} \quad [7.6]$$

These features contain enough information that a tagger should be able to choose the right tag for the word *fish*: words that come after *can* are likely to be verbs, so the feature $(w_{m-1} = \text{can}, y_m = \mathbf{V})$ should have a large positive weight.

However, even with this enhanced feature set, it may be difficult to tag some sequences correctly. One reason is that there are often relationships between the tags themselves. For example, in English it is relatively rare for a verb to follow another verb — particularly if we differentiate MODAL verbs like *can* and *should* from more typical verbs, like *give*, *transcend*, and *befuddle*. We would like to incorporate preferences against tag sequences like VERB-VERB, and in favor of tag sequences like NOUN-VERB. The need for such preferences is best illustrated by a **garden path sentence**:

(7.2) The old man the boat.

Grammatically, the word *the* is a DETERMINER. When you read the sentence, what part of speech did you first assign to *old*? Typically, this word is an ADJECTIVE — abbreviated as J — which is a class of words that modify nouns. Similarly, *man* is usually a noun. The resulting sequence of tags is D J N D N. But this is a mistaken “garden path” interpretation, 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, $V^M \times \mathcal{Y}^M \rightarrow \mathbb{R}$. 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*.

3564 In a linear model, local scoring function can be defined as a dot product of weights
 3565 and features,

$$\psi(\mathbf{w}_{1:M}, y_m, y_{m-1}, m) = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m). \quad [7.9]$$

3566 The feature vector \mathbf{f} can consider the entire input \mathbf{w} , and can look at pairs of adjacent
 3567 tags. This is a step up from per-token classification: the weights can assign low scores
 3568 to infelicitous tag pairs, such as noun-determiner, and high scores for frequent tag pairs,
 3569 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**):

$$\begin{aligned} \mathbf{f}(\mathbf{w} = \text{they can fish}, \mathbf{y} = \text{N V V}) &= \sum_{m=1}^{M+1} \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m) \\ &= \mathbf{f}(\mathbf{w}, \text{N}, \diamond, 1) \\ &\quad + \mathbf{f}(\mathbf{w}, \text{V}, \text{N}, 2) \\ &\quad + \mathbf{f}(\mathbf{w}, \text{V}, \text{V}, 3) \\ &\quad + \mathbf{f}(\mathbf{w}, \blacklozenge, \text{V}, 4) \end{aligned} \quad [7.10]$$

$$\begin{aligned} &= (w_m = \text{they}, y_m = \text{N}) + (y_m = \text{N}, y_{m-1} = \diamond) \\ &\quad + (w_m = \text{can}, y_m = \text{V}) + (y_m = \text{V}, y_{m-1} = \text{N}) \\ &\quad + (w_m = \text{fish}, y_m = \text{V}) + (y_m = \text{V}, y_{m-1} = \text{V}) \\ &\quad + (y_m = \blacklozenge, y_{m-1} = \text{V}). \end{aligned} \quad [7.11]$$

3570 There are seven active features for this example: one for each word-tag pair, and one
 3571 for each tag-tag pair, including a final tag $y_{M+1} = \blacklozenge$. These features capture the two main
 3572 sources of information for part-of-speech tagging in English: which tags are appropriate
 3573 for each word, and which tags tend to follow each other in sequence. Given appropriate
 3574 weights for these features, taggers can achieve high accuracy, even for difficult cases like
 3575 *the old man the boat*. We will now discuss how this restricted scoring function enables
 3576 efficient inference, through the **Viterbi algorithm** (Viterbi, 1967).

3577 **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:

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} \Psi(\mathbf{w}, \mathbf{y}) \quad [7.13]$$

$$= \operatorname{argmax}_{\mathbf{y}_{1:M}} \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.14]$$

$$= \operatorname{argmax}_{\mathbf{y}_{1:M}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}), \quad [7.15]$$

3578 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**, an algorithmic technique for reusing work in recurrent computations. We begin by solving an auxiliary problem: rather than finding the best tag sequence, we 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 says that we maximize over the final tag y_M , and we maximize over all “prefixes”, $\mathbf{y}_{1:M-1}$. Within the sum of scores, only the final term $s_{M+1}(\blacklozenge, y_M)$ depends on y_M , so 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  $\mathbf{y}_{1:M}$ 
```

variable called the **Viterbi variable**,

$$v_m(y_m) \triangleq \max_{\mathbf{y}_{1:m-1}} \sum_{n=1}^m s_n(y_n, y_{n-1}) \quad [7.20]$$

$$= \max_{y_{m-1}} s_m(y_m, y_{m-1}) + \max_{\mathbf{y}_{1:m-2}} \sum_{n=1}^{m-1} s_n(y_n, y_{n-1}) \quad [7.21]$$

$$= \max_{y_{m-1}} s_m(y_m, y_{m-1}) + v_{m-1}(y_{m-1}). \quad [7.22]$$

3579 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,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}_{1:M}, \mathbf{y}_{1:M}) = v_{M+1}(\blacklozenge). \quad [7.24]$$

3580 Thus, the score of the best labeling for the sequence can be computed in a single forward
 3581 sweep: first compute all variables $v_1(\cdot)$ from Equation 7.23, and then compute all variables
 3582 $v_2(\cdot)$ from the recurrence in Equation 7.22, continuing until the final variable $v_{M+1}(\blacklozenge)$.

3583 The Viterbi variables can be arranged in a structure known as a **trellis**, shown in Fig-
 3584 ure 7.1. Each column indexes a token m in the sequence, and each row indexes a tag in
 3585 \mathcal{Y} ; every $v_{m-1}(k)$ is connected to every $v_m(k')$, indicating that $v_m(k')$ is computed from
 3586 $v_{m-1}(k)$. Special nodes are set aside for the start and end states.

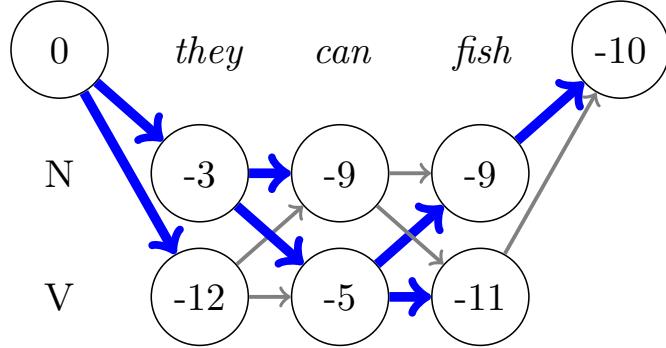


Figure 7.1: The trellis representation of the Viterbi variables, for the example *they can fish*, using the weights shown in Table 7.1.

3587 Our real goal is to find the best scoring sequence, not simply to compute its score.
 3588 But solving the auxiliary problem gets us almost all the way there. Recall that each $v_m(k)$
 3589 represents the score of the best tag sequence ending in that tag k in position m . To compute
 3590 this, we maximize over possible values of y_{m-1} . If we keep track of the “argmax” tag
 3591 that maximizes this choice at each step, then we can walk backwards from the final tag,
 3592 and recover the optimal tag sequence. This is indicated in Figure 7.1 by the thick lines,
 3593 which we trace back from the final position. These backward pointers are written $b_m(k)$,
 3594 indicating the optimal tag y_{m-1} on the path to $Y_m = k$.

3595 The complete Viterbi algorithm is shown in Algorithm 11. When computing the initial
 3596 Viterbi variables $v_1(\cdot)$, the special tag \diamond indicates the start of the sequence. When comput-
 3597 ing the final tag Y_M , another special tag, \blacklozenge indicates the end of the sequence. These special
 3598 tags enable the use of transition features for the tags that begin and end the sequence: for
 3599 example, conjunctions are unlikely to end sentences in English, so we would like a low
 3600 score for $s_{M+1}(\blacklozenge, CC)$; nouns are relatively likely to appear at the beginning of sentences,
 3601 so we would like a high score for $s_1(N, \diamond)$, assuming the noun tag is compatible with the
 3602 first word token w_1 .

3603 **Complexity** If there are K tags and M positions in the sequence, then there are $M \times K$
 3604 Viterbi variables to compute. Computing each variable requires finding a maximum over
 3605 K possible predecessor tags. The total time complexity of populating the trellis is there-
 3606 fore $\mathcal{O}(MK^2)$, with an additional factor for the number of active features at each position.
 3607 After completing the trellis, we simply trace the backwards pointers to the beginning of
 3608 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 $-\infty$.3609 **7.3.1 Example**

3610 Consider the minimal tagset $\{N, V\}$, corresponding to nouns and verbs. Even in this
 3611 tagset, there is considerable ambiguity: for example, the words *can* and *fish* can each take
 3612 both tags. Of the $2 \times 2 \times 2 = 8$ possible taggings for the sentence *they can fish*, four are
 3613 possible given these possible tags, and two are grammatical.²

3614 The values in the trellis in Figure 7.1 are computed from the feature weights defined in
 3615 Table 7.1. We begin with $v_1(N)$, which has only one possible predecessor, the start tag \diamond .
 3616 This score is therefore equal to $s_1(N, \diamond) = -2 - 1 = -3$, which is the sum of the scores for
 3617 the emission and transition features respectively; the backpointer is $b_1(N) = \diamond$. The score
 3618 for $v_1(V)$ is computed in the same way: $s_1(V, \diamond) = -10 - 2 = -12$, and again $b_1(V) = \diamond$.
 3619 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,

$$v_4(\diamond) = \max(v_3(N) + s_4(\diamond, N), v_3(V) + s_4(\diamond, V)) \quad [7.28]$$

$$= \max(-9 + 0 - 1, -11 + 0 - 1) \quad [7.29]$$

$$= -10, \quad [7.30]$$

3620 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.

3621 To compute the optimal tag sequence, we walk backwards from here, next checking
 3622 $b_3(N) = V$, and then $b_2(V) = N$, and finally $b_1(N) = \diamond$. This yields $y = (N, V, N)$, which
 3623 corresponds to the linguistic interpretation of the fishes being put into cans.

3624 **7.3.2 Higher-order features**

3625 The Viterbi algorithm was made possible by a restriction of the scoring function to local
 3626 parts that consider only pairs of adjacent tags. We can think of this as a bigram language
 3627 model over tags. A natural question is how to generalize Viterbi to tag trigrams, which
 3628 would involve the following decomposition:

$$\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+2} f(\mathbf{w}, y_m, y_{m-1}, y_{m-2}, m), \quad [7.31]$$

3629 where $y_{-1} = \diamond$ and $y_{M+2} = \blacklozenge$.

3630 One solution is to create a new tagset $\mathcal{Y}^{(2)}$ from the Cartesian product of the original
 3631 tagset with itself, $\mathcal{Y}^{(2)} = \mathcal{Y} \times \mathcal{Y}$. The tags in this product space are ordered pairs, rep-
 3632 resenting adjacent tags at the token level: for example, the tag (N, V) would represent a
 3633 noun followed by a verb. Transitions between such tags must be consistent: we can have a
 3634 transition from (N, V) to (V, N) (corresponding to the tag sequence $N V N$), but not from
 3635 (N, V) to (N, N) , which would not correspond to any coherent tag sequence. This con-
 3636 straint can be enforced in feature weights, with $\theta_{((a,b),(c,d))} = -\infty$ if $b \neq c$. The remaining
 3637 feature weights can encode preferences for and against various tag trigrams.

3638 In the Cartesian product tag space, there are K^2 tags, suggesting that the time com-
 3639 plexity will increase to $\mathcal{O}(MK^4)$. However, it is unnecessary to max over predecessor tag
 3640 bigrams that are incompatible with the current tag bigram. By exploiting this constraint,
 3641 it is possible to limit the time complexity to $\mathcal{O}(MK^3)$. The space complexity grows to
 3642 $\mathcal{O}(MK^2)$, since the trellis must store all possible predecessors of each tag. In general, the
 3643 time and space complexity of higher-order Viterbi grows exponentially with the order of
 3644 the tag n -grams that are considered in the feature decomposition.

3645 **7.4 Hidden Markov Models**

3646 The Viterbi sequence labeling algorithm is built on the scores $s_m(y, y')$. We will now
 3647 discuss how these scores can be estimated probabilistically. Recall from § 2.1 that the
 3648 probabilistic Naïve Bayes classifier selects the label y to maximize $p(y | \mathbf{x}) \propto p(y, \mathbf{x})$. In
 3649 probabilistic sequence labeling, our goal is similar: select the tag sequence that maximizes
 3650 $p(y | \mathbf{w}) \propto p(y, \mathbf{w})$. The locality restriction in Equation 7.8 can be viewed as a conditional
 3651 independence assumption on the random variables 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

```

3652 Naïve Bayes was introduced as a generative model — a probabilistic story that ex-
 3653 plains the observed data as well as the hidden label. A similar story can be constructed
 3654 for probabilistic sequence labeling: first, the tags are drawn from a prior distribution; next,
 3655 the tokens are drawn from a conditional likelihood. However, for inference to be tractable,
 3656 additional independence assumptions are required. First, the probability of each token
 3657 depends only on its tag, and not on any other element in the sequence:

$$p(w | y) = \prod_{m=1}^M p(w_m | y_m). \quad [7.32]$$

3658 Second, each tag y_m depends only on its predecessor,

$$p(y) = \prod_{m=1}^M p(y_m | y_{m-1}), \quad [7.33]$$

3659 where $y_0 = \diamond$ in all cases. Due to this **Markov assumption**, probabilistic sequence labeling
 3660 models are known as **hidden Markov models** (HMMs).

3661 The generative process for the hidden Markov model is shown in Algorithm 12. Given
 3662 the parameters λ and ϕ , we can compute $p(w, y)$ for any token sequence w and tag se-
 3663 quence y . The HMM is often represented as a **graphical model** (Wainwright and Jordan,
 3664 2008), as shown in Figure 7.2. This representation makes the independence assumptions
 3665 explicit: if a variable v_1 is probabilistically conditioned on another variable v_2 , then there
 3666 is an arrow $v_2 \rightarrow v_1$ in the diagram. If there are no arrows between v_1 and v_2 , they
 3667 are **conditionally independent**, given each variable's **Markov blanket**. In the hidden
 3668 Markov model, the Markov blanket for each tag y_m includes the “parent” y_{m-1} , and the
 3669 “children” y_{m+1} and w_m .³

3670 It is important to reflect on the implications of the HMM independence assumptions.
 3671 A non-adjacent pair of tags y_m and y_n are conditionally independent; if $m < n$ and we
 3672 are given y_{n-1} , then y_m offers no additional information about y_n . However, if we are
 3673 not given any information about the tags in a sequence, then all tags are probabilistically
 3674 coupled.

³In general graphical models, a variable's Markov blanket includes its parents, children, and its children's other parents (Murphy, 2012).

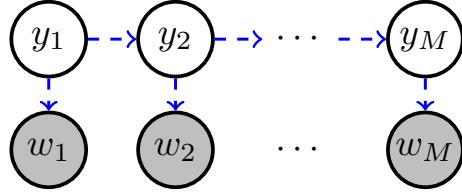


Figure 7.2: Graphical representation of the hidden Markov model. Arrows indicate probabilistic dependencies.

3675 7.4.1 Estimation

3676 The hidden Markov model has two groups of parameters:

3677 **Emission probabilities.** The probability $p_e(w_m | y_m; \phi)$ is the emission probability, since
3678 the words are treated as probabilistically “emitted”, conditioned on the tags.

3679 **Transition probabilities.** The probability $p_t(y_m | y_{m-1}; \lambda)$ is the transition probability,
3680 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,

$$\begin{aligned}\phi_{k,i} &\triangleq \Pr(W_m = i | Y_m = k) = \frac{\text{count}(W_m = i, Y_m = k)}{\text{count}(Y_m = k)} \\ \lambda_{k,k'} &\triangleq \Pr(Y_m = k' | Y_{m-1} = k) = \frac{\text{count}(Y_m = k', Y_{m-1} = k)}{\text{count}(Y_{m-1} = k)}.\end{aligned}$$

3681 Smoothing is more important for the emission probability than the transition probability,
3682 because the vocabulary is much larger than the number of tags.

3683 7.4.2 Inference

3684 The goal of inference in the hidden Markov model is to find the highest probability tag
3685 sequence,

$$\hat{y} = \underset{y}{\operatorname{argmax}} p(y | w). \quad [7.34]$$

3686 As in Naïve Bayes, it is equivalent to find the tag sequence with the highest *log*-probability,
3687 since the logarithm is a monotonically increasing function. It is furthermore equivalent
3688 to maximize the joint probability $p(y, w) = p(y | w) \times p(w) \propto p(y | w)$, which is pro-
3689 portional to the conditional probability. Putting these observations together, the inference

problem can be reformulated as,

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}} \log p(\mathbf{y}, \mathbf{w}). \quad [7.35]$$

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,

$$s_m(y_m, y_{m-1}) \triangleq \log \lambda_{y_m, y_{m-1}} + \log \phi_{y_m, w_m}, \quad [7.40]$$

and,

$$\phi_{\diamond, w} = \begin{cases} 1, & w = \blacksquare \\ 0, & \text{otherwise,} \end{cases} \quad [7.41]$$

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,

$$v_m(y_m) = \max_{\mathbf{y}_{1:m-1}} p(\mathbf{y}_{1:m-1}, y_m, \mathbf{w}_{1:m}) \quad [7.50]$$

$$= \max_{y_{m-1}} p(y_m, w_m | y_{m-1}) \times \max_{\mathbf{y}_{1:m-2}} p(\mathbf{y}_{1:m-2}, y_{m-1}, \mathbf{w}_{1:m-1}) \quad [7.51]$$

$$= \max_{y_{m-1}} p(y_m, w_m | y_{m-1}) \times v_{m-1}(y_{m-1}) \quad [7.52]$$

$$= p_{w|y}(w_m | y_m) \times \max_{y_{m-1}} p_y(y_m | y_{m-1}) \times v_{m-1}(y_{m-1}). \quad [7.53]$$

3693 Each Viterbi variable is computed by *maximizing* over a set of *products*. Thus, the Viterbi
 3694 algorithm is a special case of the **max-product algorithm** for inference in graphical mod-
 3695 els (Wainwright and Jordan, 2008). However, the product of probabilities tends towards
 3696 zero over long sequences, so the log-probability version of Viterbi is recommended in
 3697 practical implementations.

3698 7.5 Discriminative sequence labeling with features

3699 Today, hidden Markov models are rarely used for supervised sequence labeling. This is
 3700 because HMMs are limited to only two phenomena:

- 3701 • word-tag compatibility, via the emission probability $p_{W|Y}(w_m | y_m)$;
- 3702 • local context, via the transition probability $p_Y(y_m | y_{m-1})$.

3703 The Viterbi algorithm permits the inclusion of richer information in the local scoring func-
 3704 tion $\psi(\mathbf{w}_{1:M}, y_m, y_{m-1}, m)$, which can be defined as a weighted sum of arbitrary local *fea-*
 3705 *tures*,

$$\psi(\mathbf{w}, y_m, y_{m-1}, m) = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m), \quad [7.54]$$

3706 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:

$$\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.55]$$

$$= \theta \cdot \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.56]$$

$$= \theta \cdot \sum_{m=1}^{M+1} \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.57]$$

$$= \theta \cdot \mathbf{f}^{(\text{global})}(\mathbf{w}, \mathbf{y}_{1:M}), \quad [7.58]$$

3707 where $\mathbf{f}^{(\text{global})}(\mathbf{w}, \mathbf{y})$ is a global feature vector, which is a sum of local feature vectors,

$$\mathbf{f}^{(\text{global})}(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \mathbf{f}(\mathbf{w}_{1:M}, y_m, y_{m-1}, m), \quad [7.59]$$

3708 with $y_{M+1} = \diamond$ and $y_0 = \diamond$ by construction.

3709 Let's now consider what additional information these features might encode.

3710 **Word affix features.** Consider the problem of part-of-speech tagging on the first four
3711 lines of the poem *Jabberwocky* (Carroll, 1917):

3712 (7.3) 'Twas brillig, and the slithy toves
3713 Did gyre and gimble in the wabe:
3714 All mimsy were the borogoves,
3715 And the mome raths outgrabe.

3716 Many of these words were made up by the author of the poem, so a corpus would offer
3717 no information about their probabilities of being associated with any particular part of
3718 speech. Yet it is not so hard to see what their grammatical roles might be in this passage.
3719 Context helps: for example, the word *slithy* follows the determiner *the*, so it is probably a
3720 noun or adjective. Which do you think is more likely? The suffix *-thy* is found in a number
3721 of adjectives, like *frothy*, *healthy*, *pithy*, *worthy*. It is also found in a handful of nouns — e.g.,
3722 *apathy*, *sympathy* — but nearly all of these have the longer coda *-pathy*, unlike *slithy*. So the
3723 suffix gives some evidence that *slithy* is an adjective, and indeed it is: later in the text we
3724 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. chapter 9 touches on computational approaches to morphological analysis. See Bender (2013) for an overview of the underlying linguistic principles, and Haspelmath and Sims (2013) or Lieber (2015) for a full treatment.

3725 **Fine-grained context.** The hidden Markov model captures contextual information in the
 3726 form of part-of-speech tag bigrams. But sometimes, the necessary contextual information
 3727 is more specific. Consider the noun phrases *this fish* and *these fish*. Many part-of-speech
 3728 tagsets distinguish between singular and plural nouns, but do not distinguish between
 3729 singular and plural determiners; for example, the well known **Penn Treebank** tagset fol-
 3730 lows these conventions. A hidden Markov model would be unable to correctly label *fish* as
 3731 singular or plural in both of these cases, because it only has access to two features: the pre-
 3732 ceding tag (determiner in both cases) and the word (*fish* in both cases). The classification-
 3733 based tagger discussed in § 7.1 had the ability to use preceding and succeeding words as
 3734 features, and it can also be incorporated into a Viterbi-based sequence labeler as a local
 3735 feature.

Example. Consider the tagging D J N (determiner, adjective, noun) for the sequence *the slithy toves*, so that

$$\begin{aligned} \mathbf{w} &= \text{the slithy toves} \\ \mathbf{y} &= \text{D J N}. \end{aligned}$$

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.⁵ The resulting feature vector is,

$$\begin{aligned} \mathbf{f}(\text{the slithy toves, D J N}) &= \mathbf{f}(\text{the slithy toves, D}, \diamond, 1) \\ &\quad + \mathbf{f}(\text{the slithy toves, J}, \text{D}, 2) \\ &\quad + \mathbf{f}(\text{the slithy toves, N}, \text{J}, 3) \\ &\quad + \mathbf{f}(\text{the slithy toves}, \blacklozenge, \text{N}, 4) \\ &= \{(T : \diamond, \text{D}), (W : \text{the}, \text{D}), (M : \emptyset, \text{D}), \\ &\quad (T : \text{D}, \text{J}), (W : \text{slithy}, \text{J}), (M : \text{-thy}, \text{J}), \\ &\quad (T : \text{J}, \text{N}), (W : \text{toves}, \text{N}), (M : \text{-es}, \text{N}) \\ &\quad (T : \text{N}, \blacklozenge)\}. \end{aligned}$$

3736 These examples show that local features can incorporate information that lies beyond
 3737 the scope of a hidden Markov model. Because the features are local, it is possible to apply
 3738 the Viterbi algorithm to identify the optimal sequence of tags. The remaining question

⁵Such 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*.

3739 is how to estimate the weights on these features. § 2.2 presented three main types of
 3740 discriminative classifiers: perceptron, support vector machine, and logistic regression.
 3741 Each of these classifiers has a structured equivalent, enabling it to be trained from labeled
 3742 sequences rather than individual tokens.

3743 **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,

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} \theta \cdot f(\mathbf{w}, \mathbf{y}) \quad [7.62]$$

$$\theta^{(t+1)} \leftarrow \theta^{(t)} + f(\mathbf{w}, \mathbf{y}) - f(\mathbf{w}, \hat{\mathbf{y}}). \quad [7.63]$$

3744 This learning algorithm is called **structured perceptron**, because it learns to predict the
 3745 structured output \mathbf{y} . The only difference is that instead of computing \hat{y} by enumerating
 3746 the entire set \mathcal{Y} , the Viterbi algorithm is used to efficiently search the set of possible tag-
 3747 gings, \mathcal{Y}^M . Structured perceptron can be applied to other structured outputs as long as
 3748 efficient inference is possible. As in perceptron classification, weight averaging is crucial
 3749 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:

$$\theta_{(fish,V)} \leftarrow \theta_{(fish,V)} + 1, \quad \theta_{(fish,N)} \leftarrow \theta_{(fish,N)} - 1 \quad [7.64]$$

$$\theta_{(V,V)} \leftarrow \theta_{(V,V)} + 1, \quad \theta_{(V,N)} \leftarrow \theta_{(V,N)} - 1 \quad [7.65]$$

$$\theta_{(V,\blacklozenge)} \leftarrow \theta_{(V,\blacklozenge)} + 1, \quad \theta_{(N,\blacklozenge)} \leftarrow \theta_{(N,\blacklozenge)} - 1. \quad [7.66]$$

3750 **7.5.2 Structured support vector machines**

3751 Large-margin classifiers such as the support vector machine improve on the perceptron by
 3752 pushing the classification boundary away from the training instances. The same idea can

3753 be applied to sequence labeling. A support vector machine in which the output is a struc-
 3754 tured object, such as a sequence, is called a **structured support vector machine** (Tsochan-
 3755 taridis et al., 2004).⁶

3756 In classification, we formalized the large-margin constraint as,

$$\forall \mathbf{y} \neq \mathbf{y}^{(i)}, \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}^{(i)}) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}) \geq 1, \quad [7.67]$$

3757 requiring a margin of at least 1 between the scores for all labels \mathbf{y} that are not equal to the
 3758 correct label $\mathbf{y}^{(i)}$. The weights $\boldsymbol{\theta}$ are then learned by constrained optimization (see § 2.3.2).

3759 This idea can be applied to sequence labeling by formulating an equivalent set of con-
 3760 straints for all possible labelings $\mathcal{Y}(\mathbf{w})$ for an input \mathbf{w} . However, there are two problems.
 3761 First, in sequence labeling, some predictions are more wrong than others: we may miss
 3762 only one tag out of fifty, or we may get all fifty wrong. We would like our learning algo-
 3763 rithm to be sensitive to this difference. Second, the number of constraints is equal to the
 3764 number of possible labelings, which is exponentially large in the length of the sequence.

3765 The first problem can be addressed by adjusting the constraint to require larger mar-
 3766 gins for more serious errors. Let $c(\mathbf{y}^{(i)}, \hat{\mathbf{y}}) \geq 0$ represent the *cost* of predicting label $\hat{\mathbf{y}}$ when
 3767 the true label is $\mathbf{y}^{(i)}$. We can then generalize the margin constraint,

$$\forall \mathbf{y}, \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) \geq c(\mathbf{y}^{(i)}, \mathbf{y}). \quad [7.68]$$

3768 This cost-augmented margin constraint specializes to the constraint in Equation 7.67 if we
 3769 choose the delta function $c(\mathbf{y}^{(i)}, \mathbf{y}) = \delta((\mathbf{y}^{(i)} \neq \mathbf{y}))$. A more expressive cost function is
 3770 the **Hamming cost**,

$$c(\mathbf{y}^{(i)}, \mathbf{y}) = \sum_{m=1}^M \delta(y_m^{(i)} \neq y_m), \quad [7.69]$$

3771 which computes the number of errors in \mathbf{y} . By incorporating the cost function as the
 3772 margin constraint, we require that the true labeling be separated from the alternatives by
 3773 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:

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \neq \mathbf{y}^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) + c(\mathbf{y}^{(i)}, \mathbf{y}) \quad [7.70]$$

$$= \operatorname{argmax}_{\mathbf{y} \neq \mathbf{y}^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) + c(\mathbf{y}^{(i)}, \mathbf{y}), \quad [7.71]$$

⁶This 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.

3774 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,

$$\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) - \max_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} (\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}) + c(\mathbf{y}^{(i)}, \mathbf{y})) \geq 0. \quad [7.72]$$

3775 If the score for $\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})$ is greater than the cost-augmented score for all alternatives,
 3776 then the constraint will be met. The name “cost-augmented decoding” is due to the fact
 3777 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
 3778 an additional term for the cost. Essentially, we want to train against predictions that are
 3779 strong and wrong: they should score highly according to the model, yet incur a large loss
 3780 with respect to the ground truth. Training adjusts the weights to reduce the score of these
 3781 predictions.

3782 For cost-augmented decoding to be tractable, the cost function must decompose into
 3783 local parts, just as the feature function $f(\cdot)$ does. The Hamming cost, defined above,
 3784 obeys this property. To perform cost-augmented decoding using the Hamming cost, we
 3785 need only to add features $f_m(y_m) = \delta(y_m \neq y_m^{(i)})$, and assign a constant weight of 1 to
 3786 these features. Decoding can then be performed using the Viterbi algorithm.⁷

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:

$$\min_{\theta} \frac{1}{2} \|\theta\|_2^2 - C \left(\sum_i \theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) - \max_{\mathbf{y} \in \mathcal{Y}(\mathbf{w}^{(i)})} [\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}) + c(\mathbf{y}^{(i)}, \mathbf{y})] \right), \quad [7.73]$$

3787 In this formulation, C is a parameter that controls the tradeoff between the regularization
 3788 term and the margin constraints. A number of optimization algorithms have been
 3789 proposed for structured support vector machines, some of which are discussed in § 2.3.2.
 3790 An empirical comparison by Kummerfeld et al. (2015) shows that stochastic subgradient
 3791 descent — which is essentially a cost-augmented version of the structured perceptron —
 3792 is highly competitive.

3793 7.5.3 Conditional random fields

3794 The **conditional random field** (CRF; Lafferty et al., 2001) is a conditional probabilistic
 3795 model for sequence labeling; just as structured perceptron is built on the perceptron clas-
 3796 sifier, conditional random fields are built on the logistic regression classifier.⁸ The basic

⁷Are 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.

⁸The 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

3797 probability model is,

$$p(\mathbf{y} \mid \mathbf{w}) = \frac{\exp(\Psi(\mathbf{w}, \mathbf{y}))}{\sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp(\Psi(\mathbf{w}, \mathbf{y}'))}. \quad [7.74]$$

3798 This is almost identical to logistic regression, but because the label space is now tag
 3799 sequences, we require efficient algorithms for both **decoding** (searching for the best tag
 3800 sequence given a sequence of words \mathbf{w} and a model θ) and for **normalizing** (summing
 3801 over all tag sequences). These algorithms will be based on the usual locality assumption
 3802 on the scoring function, $\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m)$.

3803 **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 $p(\mathbf{y} \mid \mathbf{w})$,

$$\begin{aligned} \hat{\mathbf{y}} &= \operatorname{argmax}_{\mathbf{y}} \log p(\mathbf{y} \mid \mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{y}} \Psi(\mathbf{y}, \mathbf{w}) - \log \sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp \Psi(\mathbf{y}', \mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{y}} \Psi(\mathbf{y}, \mathbf{w}) = \operatorname{argmax}_{\mathbf{y}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}). \end{aligned}$$

3804 This is identical to the decoding problem for structured perceptron, so the same Viterbi
 3805 recurrence as defined in Equation 7.22 can be used.

3806 **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.

3807 where λ controls the amount of regularization. The final term in Equation 7.76 is a sum
 3808 over all possible labelings. This term is the log of the denominator in Equation 7.74, some-
 3809 times known as the **partition function**.⁹ There are $|\mathcal{Y}|^M$ possible labelings of an input of
 3810 size M , so we must again exploit the decomposition of the scoring function to compute
 3811 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,

$$\sum_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} \Psi(\mathbf{w}, \mathbf{y}) = \sum_{\mathbf{y}_{1:M}} s_{M+1}(\blacklozenge, y_M) \prod_{m=1}^M s_m(y_m, y_{m-1}) \quad [7.82]$$

$$= \alpha_{M+1}(\blacklozenge). \quad [7.83]$$

⁹The terminology of “potentials” and “partition functions” comes from statistical mechanics (Bishop, 2006).

The conditional log-likelihood can be rewritten,

$$\ell = \frac{\lambda}{2} \|\boldsymbol{\theta}\|^2 - \sum_{i=1}^N \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) + \log \alpha_{M+1}(\blacklozenge). \quad [7.84]$$

- 3812 Probabilistic programming environments, such as `Torch` (Collobert et al., 2011) and `dynet` (Neu-
 3813 big et al., 2017), can compute the gradient of this objective using automatic differentiation.
 3814 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:

$$\frac{d\ell}{d\theta_j} = \lambda \theta_j + \sum_{i=1}^N E[f_j(\mathbf{w}^{(i)}, \mathbf{y})] - f_j(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}), \quad [7.85]$$

- 3815 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-
 3816 quence $\mathbf{y}^{(i)}$. The expected feature counts are computed “under the hood” when automatic
 3817 differentiation is applied to Equation 7.84 (Eisner, 2016).

- 3818 Before the widespread use of automatic differentiation, it was common to compute
 3819 the feature expectations from marginal tag probabilities $p(y_m | \mathbf{w})$. These marginal prob-
 3820 abilities are sometimes useful on their own, and can be computed using the **forward-**
 3821 **backward algorithm**. This algorithm combines the forward recurrence with an equivalent
 3822 **backward recurrence**, which traverses the input from w_M back to w_1 .

3823 7.5.3.3 *Forward-backward algorithm

Marginal probabilities over tag bigrams can be written as,¹⁰

$$\Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}) = \frac{\sum_{\mathbf{y}: Y_m=k, Y_{m-1}=k'} \prod_{n=1}^M \exp s_n(y_n, y_{n-1})}{\sum_{\mathbf{y}'} \prod_{n=1}^M \exp s_n(y'_n, y'_{n-1})}. \quad [7.86]$$

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

¹⁰Recall 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 .

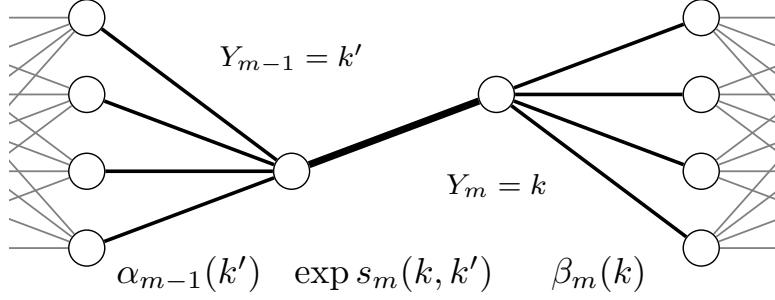


Figure 7.3: A schematic illustration of the computation of the marginal probability $\Pr(Y_{m-1} = k', Y_m = k)$, using the forward score $\alpha_{m-1}(k')$ and the backward score $\beta_m(k)$.

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**:

$$\beta_m(k) \triangleq \sum_{\mathbf{y}_{m:M}: Y_m = k} \prod_{n=m}^{M+1} \exp s_n(y_n, y_{n-1}) \quad [7.88]$$

$$= \sum_{k' \in \mathcal{Y}} \exp s_{m+1}(k', k) \sum_{\mathbf{y}_{m+1:M}: Y_m = k'} \prod_{n=m+1}^{M+1} \exp s_n(y_n, y_{n-1}) \quad [7.89]$$

$$= \sum_{k' \in \mathcal{Y}} \exp s_{m+1}(k', k) \times \beta_{m+1}(k'). \quad [7.90]$$

³⁸²⁴ 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,

$$p(y | \mathbf{w}_{1:m}) = \frac{\exp \psi_m(y)}{\sum_{y' \in \mathcal{Y}} \exp \psi_m(y')}. \quad [7.95]$$

3843 Using this transformation, it is possible to train the tagger from the negative log-likelihood
 3844 of the tags, as in a conditional random field. Alternatively, a hinge loss or margin loss
 3845 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:

$$\overleftarrow{\mathbf{h}}_m = g(\mathbf{x}_m, \overleftarrow{\mathbf{h}}_{m+1}), \quad m = 1, 2, \dots, M. \quad [7.96]$$

3846 The hidden states of the left-to-right RNN are denoted $\overrightarrow{\mathbf{h}}_m$. The left-to-right and right-to-
 3847 left vectors are concatenated, $\mathbf{h}_m = [\overleftarrow{\mathbf{h}}_m; \overrightarrow{\mathbf{h}}_m]$. The scoring function in Equation 7.93 is
 3848 applied to this concatenated vector.

3849 Bidirectional RNN tagging has several attractive properties. Ideally, the representa-
 3850 tion \mathbf{h}_m summarizes the useful information from the surrounding context, so that it is not
 3851 necessary to design explicit features to capture this information. If the vector \mathbf{h}_m is an ad-
 3852 equate summary of this context, then it may not even be necessary to perform the tagging
 3853 jointly: in general, the gains offered by joint tagging of the entire sequence are diminished
 3854 as the individual tagging model becomes more powerful. Using backpropagation, the
 3855 word vectors \mathbf{x} can be trained “end-to-end”, so that they capture word properties that are
 3856 useful for the tagging task. Alternatively, if limited labeled data is available, we can use
 3857 word embeddings that are “pre-trained” from unlabeled data, using a language modeling
 3858 objective (as in § 6.3) or a related word embedding technique (see chapter 14). It is even
 3859 possible to combine both fine-tuned and pre-trained embeddings in a single model.

3860 **Neural structure prediction** The bidirectional recurrent neural network incorporates in-
 3861 formation from throughout the input, but each tagging decision is made independently.
 3862 In some sequence labeling applications, there are very strong dependencies between tags:
 3863 it may even be impossible for one tag to follow another. In such scenarios, the tagging
 3864 decision must be made jointly across the entire sequence.

3865 Neural sequence labeling can be combined with the Viterbi algorithm by defining the
 3866 local scores as:

$$s_m(y_m, y_{m-1}) = \beta_{y_m} \cdot \mathbf{h}_m + \eta_{y_{m-1}, y_m}, \quad [7.97]$$

3867 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}
 3868 is a scalar parameter for the tag transition (y_{m-1}, y_m) . These local scores can then be
 3869 incorporated into the Viterbi algorithm for inference, and into the forward algorithm for
 3870 training. This model is shown in Figure 7.4. It can be trained from the conditional log-
 3871 likelihood objective defined in Equation 7.76, backpropagating to the tagging parameters

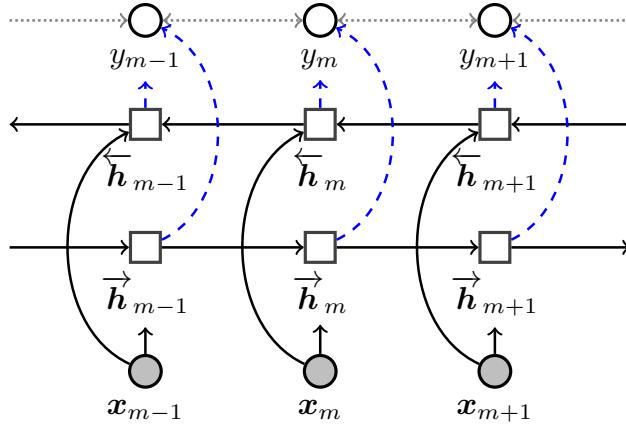


Figure 7.4: Bidirectional LSTM for sequence labeling. The solid lines indicate computation, the dashed lines indicate probabilistic dependency, and the dotted lines indicate the optional additional probabilistic dependencies between labels in the biLSTM-CRF.

3872 β and η , as well as the parameters of the RNN. This model is called the **LSTM-CRF**, due
 3873 to its combination of aspects of the long short-term memory and conditional random field
 3874 models (Huang et al., 2015).

3875 The LSTM-CRF is especially effective on the task of **named entity recognition** (Lample
 3876 et al., 2016), a sequence labeling task that is described in detail in § 8.3. This task has strong
 3877 dependencies between adjacent tags, so structure prediction is especially important.

3878 7.6.2 Character-level models

3879 As in language modeling, rare and unseen words are a challenge: if we encounter a word
 3880 that was not in the training data, then there is no obvious choice for the word embed-
 3881 ding x_m . One solution is to use a generic **unseen word** embedding for all such words.
 3882 However, in many cases, properties of unseen words can be guessed from their spellings.
 3883 For example, *whimsical* does not appear in the Universal Dependencies (UD) English Tree-
 3884 bank, yet the suffix *-al* makes it likely to be adjective; by the same logic, *unflinchingly* is
 3885 likely to be an adverb, and *barnacle* is likely to be a noun.

3886 In feature-based models, these morphological properties were handled by suffix fea-
 3887 tures; in a neural network, they can be incorporated by constructing the embeddings of
 3888 unseen words from their spellings or morphology. One way to do this is to incorporate
 3889 an additional layer of bidirectional RNNs, one for each word in the vocabulary (Ling
 3890 et al., 2015). For each such character-RNN, the inputs are the characters, and the output
 3891 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:

$$\Pr(W = i \mid Y = k) = \phi_{k,i} = \frac{E[\text{count}(W = i, Y = k)]}{E[\text{count}(Y = k)]}$$

$$\Pr(Y_m = k \mid Y_{m-1} = k') = \lambda_{k',k} = \frac{E[\text{count}(Y_m = k, Y_{m-1} = k')]}{E[\text{count}(Y_{m-1} = k')]} \quad 3917$$

3918 The expected counts are computed in the E-step, using the forward and backward
 3919 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,

$$\Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}) = \frac{\alpha_{m-1}(k') \times \exp s_m(k, k') \times \beta_m(k)}{\alpha_{M+1}(\blacklozenge)}. \quad [7.101]$$

In a hidden Markov model, each element of the forward-backward computation has a special interpretation:

$$\alpha_{m-1}(k') = p(Y_{m-1} = k', \mathbf{w}_{1:m-1}) \quad [7.102]$$

$$\exp s_m(k, k') = p(Y_m = k, w_m | Y_{m-1} = k') \quad [7.103]$$

$$\beta_m(k) = p(\mathbf{w}_{m+1:M} | Y_m = k). \quad [7.104]$$

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,

$$\begin{aligned} \alpha_{m-1}(k') \times \exp s_m(k, k') \times \beta_m(k) &= p(Y_{m-1} = k', \mathbf{w}_{1:m-1}) \\ &\quad \times p(Y_m = k, w_m | Y_{m-1} = k') \\ &\quad \times p(\mathbf{w}_{m+1:M} | Y_m = k) \\ &= p(Y_{m-1} = k', Y_m = k, \mathbf{w}_{1:M}). \end{aligned} \quad [7.105]$$

Dividing by $\alpha_{M+1}(\blacklozenge) = p(\mathbf{w}_{1:M})$ gives the desired probability,

$$\frac{\alpha_{m-1}(k') \times s_m(k, k') \times \beta_m(k)}{\alpha_{M+1}(\blacklozenge)} = \frac{p(Y_{m-1} = k', Y_m = k, \mathbf{w}_{1:M})}{p(\mathbf{w}_{1:M})} \quad [7.106]$$

$$= \Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}_{1:M}). \quad [7.107]$$

3920 The expected emission counts can be computed in a similar manner, using the product
 3921 $\alpha_m(k) \times \beta_m(k)$.

3922 **7.7.1 Linear dynamical systems**

3923 The forward-backward algorithm can be viewed as Bayesian state estimation in a discrete
 3924 state space. In a continuous state space, $\mathbf{y}_m \in \mathbb{R}^K$, the equivalent algorithm is the **Kalman**
 3925 **smoother**. It also computes marginals $p(\mathbf{y}_m | \mathbf{x}_{1:M})$, using a similar two-step algorithm
 3926 of forward and backward passes. Instead of computing a trellis of values at each step, the
 3927 Kalman smoother computes a probability density function $q_{\mathbf{y}_m}(\mathbf{y}_m; \boldsymbol{\mu}_m, \Sigma_m)$, character-
 3928 ized by a mean $\boldsymbol{\mu}_m$ and a covariance Σ_m around the latent state. Connections between the
 3929 Kalman smoother and the forward-backward algorithm are elucidated by Minka (1999)
 3930 and Murphy (2012).

3931 **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):

$$p(y_m | \mathbf{y}_{-m}, \mathbf{w}_{1:M}) \propto p(w_m | y_m) p(y_m | \mathbf{y}_{-m}). \quad [7.108]$$

3932 Gibbs Sampling has been applied to unsupervised part-of-speech tagging by Goldwater
 3933 and Griffiths (2007). **Beam sampling** is a more sophisticated sampling algorithm, which
 3934 randomly draws entire sequences $\mathbf{y}_{1:M}$, rather than individual tags y_m ; this algorithm
 3935 was applied to unsupervised part-of-speech tagging by Van Gael et al. (2009). Spectral
 3936 learning (see § 5.5.2) can also be applied to sequence labeling. By factoring matrices of
 3937 co-occurrence counts of word bigrams and trigrams (Song et al., 2010; Hsu et al., 2012), it
 3938 is possible to obtain globally optimal estimates of the transition and emission parameters,
 3939 under mild assumptions.

3940 **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 symbols \oplus and \otimes represent generalized addition and multiplication respectively.¹¹ Given these operators, a generalized Viterbi recurrence is denoted,

$$v_m(k) = \bigoplus_{k' \in \mathcal{Y}} s_m(k, k') \otimes v_{m-1}(k'). \quad [7.109]$$

¹¹In 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 $a \otimes \bar{0} = \bar{0}$.

3941 Each recurrence that we have seen so far is a special case of this generalized Viterbi
 3942 recurrence:

- 3943 • In the max-product Viterbi recurrence over probabilities, the \oplus operation corre-
 3944 sponds to maximization, and the \otimes operation corresponds to multiplication.
- 3945 • In the forward recurrence over probabilities, the \oplus operation corresponds to addition,
 3946 and the \otimes operation corresponds to multiplication.
- 3947 • In the max-product Viterbi recurrence over log-probabilities, the \oplus operation corre-
 3948 sponds to maximization, and the \otimes operation corresponds to addition.¹²
- 3949 • In the forward recurrence over log-probabilities, the \oplus operation corresponds to log-
 3950 addition, $a \oplus b = \log(e^a + e^b)$. The \otimes operation corresponds to addition.

3951 The mathematical abstraction offered by semiring notation can be applied to the soft-
 3952 ware implementations of these algorithms, yielding concise and modular implemen-
 3953 tations. For example, in the OPENFST library, generic operations are parametrized by the
 3954 choice of semiring (Allauzen et al., 2007).

3955 Exercises

- 3956 1. Extend the example in § 7.3.1 to the sentence *they can can fish*, meaning that “they can
 3957 put fish into cans.” Build the trellis for this example using the weights in Table 7.1,
 3958 and identify the best-scoring tag sequence. If the scores for noun and verb are tied,
 3959 then you may assume that the backpointer always goes to noun.
- 3960 2. Using the tagset $\mathcal{Y} = \{N, V\}$, and the feature set $f(\mathbf{w}, y_m, y_{m-1}, m) = \{(w_m, y_m), (y_m, y_{m-1})\}$,
 3961 show that there is no set of weights that give the correct tagging for both *they can*
 3962 *fish* (N V V) and *they can can fish* (N V V N).
- 3963 3. Work out what happens if you train a structured perceptron on the two exam-
 3964 ples mentioned in the previous problem, using the transition and emission features
 3965 (y_m, y_{m-1}) and (y_m, w_m) . Initialize all weights at 0, and assume that the Viterbi algo-
 3966 rithm always chooses *N* when the scores for the two tags are tied, so that the initial
 3967 prediction for *they can fish* is N N N.
- 3968 4. Consider the garden path sentence, *The old man the boat*. Given word-tag and tag-tag
 3969 features, what inequality in the weights must hold for the correct tag sequence to
 3970 outscore the garden path tag sequence for this example?

¹²This is sometimes called the **tropical semiring**, in honor of the Brazilian mathematician Imre Simon.

- 3971 5. Using the weights in Table 7.1, explicitly compute the log-probabilities for all pos-
 3972 sible taggings of the input *fish can*. Verify that the forward algorithm recovers the
 3973 aggregate log probability.
- 3974 6. Sketch out an algorithm for a variant of Viterbi that returns the top-*n* label se-
 3975 quences. What is the time and space complexity of this algorithm?
- 3976 7. Show how to compute the marginal probability $\Pr(y_{m-2} = k, y_m = k' \mid \mathbf{w}_{1:M})$, in
 3977 terms of the forward and backward variables, and the potentials $s_n(y_n, y_{n-1})$.
- 3978 8. Suppose you receive a stream of text, where some of tokens have been replaced at
 3979 random with *NOISE*. For example:
- 3980 • Source: *I try all things, I achieve what I can*
 - 3981 • Message received: *I try NOISE NOISE, I NOISE what I NOISE*
- 3982 Assume you have access to a pre-trained bigram language model, which gives prob-
 3983 abilities $p(w_m \mid w_{m-1})$. These probabilities can be assumed to be non-zero for all
 3984 bigrams.
- 3985 Show how to use the Viterbi algorithm to recover the source by maximizing the
 3986 bigram language model log-probability. Specifically, set the scores $s_m(y_m, y_{m-1})$ so
 3987 that the Viterbi algorithm selects a sequence of words that maximizes the bigram
 3988 language model log-probability, while leaving the non-noise tokens intact. Your
 3989 solution should not modify the logic of the Viterbi algorithm, it should only set the
 3990 scores $s_m(y_m, y_{m-1})$.
- 3991 9. Let $\alpha(\cdot)$ and $\beta(\cdot)$ indicate the forward and backward variables as defined in § 7.5.3.3.
 3992 Prove that $\alpha_{M+1}(\blacklozenge) = \beta_0(\lozenge) = \sum_y \alpha_m(y) \beta_m(y), \forall m \in \{1, 2, \dots, M\}$.
- 3993 10. Consider an RNN tagging model with a tanh activation function on the hidden
 3994 layer, and a hinge loss on the output. (The problem also works for the margin loss
 3995 and negative log-likelihood.) Suppose you initialize all parameters to zero: this in-
 3996 cludes the word embeddings that make up \mathbf{x} , the transition matrix Θ , the output
 3997 weights β , and the initial hidden state \mathbf{h}_0 .
 - 3998 a) Prove that for any data and for any gradient-based learning algorithm, all pa-
 3999 rameters will be stuck at zero.
 - 4000 b) Would a sigmoid activation function avoid this problem?

4001 Chapter 8

4002 Applications of sequence labeling

4003 Sequence labeling has applications throughout natural language processing. This chap-
4004 ter focuses on part-of-speech tagging, morpho-syntactic attribute tagging, named entity
4005 recognition, and tokenization. It also touches briefly on two applications to interactive
4006 settings: dialogue act recognition and the detection of code-switching points between
4007 languages.

4008 8.1 Part-of-speech tagging

4009 The **syntax** of a language is the set of principles under which sequences of words are
4010 judged to be grammatically acceptable by fluent speakers. One of the most basic syntactic
4011 concepts is the **part-of-speech** (POS), which refers to the syntactic role of each word in a
4012 sentence. This concept was used informally in the previous chapter, and you may have
4013 some intuitions from your own study of English. For example, in the sentence *We like*
4014 *vegetarian sandwiches*, you may already know that *we* and *sandwiches* are nouns, *like* is a
4015 verb, and *vegetarian* is an adjective. These labels depend on the context in which the word
4016 appears: in *she eats like a vegetarian*, the word *like* is a preposition, and the word *vegetarian*
4017 is a noun.

4018 Parts-of-speech can help to disentangle or explain various linguistic problems. Recall
4019 Chomsky's proposed distinction in chapter 6:

- 4020 (8.1) Colorless green ideas sleep furiously.
- 4021 (8.2) *Ideas colorless furiously green sleep.

4022 One difference between these two examples is that the first contains part-of-speech transitions
4023 that are typical in English: adjective to adjective, adjective to noun, noun to verb, and verb
4024 to adverb. The second example contains transitions that are unusual: noun to adjective
4025 and adjective to verb. The ambiguity in a headline like,

4026 (8.3) Teacher Strikes Idle Children

4027 can also be explained in terms of parts of speech: in the interpretation that was likely
 4028 intended, *strikes* is a noun and *idle* is a verb; in the alternative explanation, *strikes* is a verb
 4029 and *idle* is an adjective.

4030 Part-of-speech tagging is often taken as a early step in a natural language processing
 4031 pipeline. Indeed, parts-of-speech provide features that can be useful for many of the
 4032 tasks that we will encounter later, such as parsing (chapter 10), coreference resolution
 4033 (chapter 15), and relation extraction (chapter 17).

4034 **8.1.1 Parts-of-Speech**

4035 The **Universal Dependencies** project (UD) is an effort to create syntactically-annotated
 4036 corpora across many languages, using a single annotation standard (Nivre et al., 2016). As
 4037 part of this effort, they have designed a part-of-speech **tagset**, which is meant to capture
 4038 word classes across as many languages as possible.¹ This section describes that inventory,
 4039 giving rough definitions for each of tags, along with supporting examples.

4040 Part-of-speech tags are **morphosyntactic**, rather than **semantic**, categories. This means
 4041 that they describe words in terms of how they pattern together and how they are inter-
 4042 nally constructed (e.g., what suffixes and prefixes they include). For example, you may
 4043 think of a noun as referring to objects or concepts, and verbs as referring to actions or
 4044 events. But events can also be nouns:

4045 (8.4) ... the **howling** of the **shrieking** storm.

4046 Here *howling* and *shrieking* are events, but grammatically they act as a noun and adjective
 4047 respectively.

4048 **8.1.1.1 The Universal Dependency part-of-speech tagset**

4049 The UD tagset is broken up into three groups: open class tags, closed class tags, and
 4050 “others.”

4051 **Open class tags** Nearly all languages contain nouns, verbs, adjectives, and adverbs.²
 4052 These are all **open word classes**, because new words can easily be added to them. The
 4053 UD tagset includes two other tags that are open classes: proper nouns and interjections.

4054 • **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).

4055 (8.5) **Toes** are scarce among veteran **blubber men**.

4056 In English, nouns tend to follow determiners and adjectives, and can play the subject
4057 role in the sentence. They can be marked for the plural number by an -s suffix.

- 4058 • **Proper nouns** (PROPN) are tokens in names, which uniquely specify a given entity,

4059 (8.6) “**Moby Dick?**” shouted **Ahab**.

- 4060 • **Verbs** (VERB), according to the UD guidelines, “typically signal events and ac-
4061 tions.” But they are also defined grammatically: they “can constitute a minimal
4062 predicate in a clause, and govern the number and types of other constituents which
4063 may occur in a clause.”³

4064 (8.7) “**Moby Dick?**” shouted Ahab.

4065 (8.8) Shall we **keep chasing** this murderous fish?

4066 English verbs tend to come in between the subject and some number of direct ob-
4067 jects, depending on the verb. They can be marked for **tense** and **aspect** using suffixes
4068 such as *-ed* and *-ing*. (These suffixes are an example of **inflectional morphology**,
4069 which is discussed in more detail in § 9.1.4.)

- 4070 • **Adjectives** (ADJ) describe properties of entities,

4071 (8.9) Shall we keep chasing this **murderous** fish?

4072 (8.10) Toes are **scarce** among **veteran** blubber men.

4073 In the second example, *scarce* is a predicative adjective, linked to the subject by the
4074 **copula verb** *are*. In contrast, *murderous* and *veteran* are attributive adjectives, modi-
4075 fying the noun phrase in which they are embedded.

- 4076 • **Adverbs** (ADV) describe properties of events, and may also modify adjectives or
4077 other adverbs:

4078 (8.11) It is not down on any map; true places **never** are.

4079 (8.12) ... **treacherously** hidden beneath the loveliest tints of azure

4080 (8.13) Not drowned **entirely**, though.

- 4081 • **Interjections** (INTJ) are used in exclamations, e.g.,

4082 (8.14) **Aye aye!** it was that accursed white whale that razed me.

³<http://universaldependencies.org/u/pos/VERB.html>

4083 **Closed class tags** Closed word classes rarely receive new members. They are sometimes
 4084 referred to as **function words** — as opposed to **content words** — as they have little lexical
 4085 meaning of their own, but rather, help to organize the components of the sentence.

- 4086 • **Adpositions** (ADP) describe the relationship between a complement (usually a noun
 4087 phrase) and another unit in the sentence, typically a noun or verb phrase.

4088 (8.15) Toes are scarce **among** veteran blubber men.

4089 (8.16) It is not **down on** any map.

4090 (8.17) Give not thyself **up** then.

4091 As the examples show, English generally uses prepositions, which are adpositions
 4092 that appear before their complement. (An exception is *ago*, as in, *we met three days*
 4093 *ago*). Postpositions are used in other languages, such as Japanese and Turkish.

- 4094 • **Auxiliary verbs** (AUX) are a closed class of verbs that add information such as
 4095 tense, aspect, person, and number.

4096 (8.18) **Shall** we keep chasing this murderous fish?

4097 (8.19) What the white whale was to Ahab, **has been** hinted.

4098 (8.20) Ahab **must** use tools.

4099 (8.21) Meditation and water **are** wedded forever.

4100 (8.22) Toes **are** scarce among veteran blubber men.

4101 The final example is a copula verb, which is also tagged as an auxiliary in the UD
 4102 corpus.

- 4103 • **Coordinating conjunctions** (CCONJ) express relationships between two words or
 4104 phrases, which play a parallel role:

4105 (8.23) Meditation **and** water are wedded forever.

- 4106 • **Subordinating conjunctions** (SCONJ) link two clauses, making one syntactically
 4107 subordinate to the other:

4108 (8.24) It is the easiest thing in the world for a man to look as **if** he had a great
 4109 secret in him.

4110 Note that

- 4111 • **Pronouns** (PRON) are words that substitute for nouns or noun phrases.

4112 (8.25) Be **it what it will**, I'll go to **it** laughing.

4113 (8.26) I try all things, I achieve **what** I can.

4114 The example includes the personal pronouns *I* and *it*, as well as the relative pronoun
 4115 *what*. Other pronouns include *myself*, *somebody*, and *nothing*.

- 4116 • **Determiners** (DET) provide additional information about the nouns or noun phrases
 4117 that they modify:

4118 (8.27) What **the** white whale was to Ahab, has been hinted.

4119 (8.28) It is not down on **any** map.

4120 (8.29) I try **all** things ...

4121 (8.30) Shall we keep chasing **this** murderous fish?

4122 Determiners include articles (*the*), possessive determiners (*their*), demonstratives
 4123 (*this murderous fish*), and quantifiers (*any map*).

- 4124 • **Numerals** (NUM) are an infinite but closed class, which includes integers, fractions,
 4125 and decimals, regardless of whether spelled out or written in numerical form.

4126 (8.31) How then can this **one** small heart beat.

4127 (8.32) I am going to put him down for the **three hundredth**.

- 4128 • **Particles** (PART) are a catch-all of function words that combine with other words or
 4129 phrases, but do not meet the conditions of the other tags. In English, this includes
 4130 the infinitival *to*, the possessive marker, and negation.

4131 (8.33) Better **to** sleep with a sober cannibal than a drunk Christian.

4132 (8.34) So man's insanity is heaven's sense

4133 (8.35) It is **not** down on any map

4134 As the second example shows, the possessive marker is not considered part of the
 4135 same token as the word that it modifies, so that *man's* is split into two tokens. (Tok-
 4136 enization is described in more detail in § 8.4.) A non-English example of a particle
 4137 is the Japanese question marker *ka*:⁴

4138 (8.36) Sensei desu ka

Teacher is ?

4139 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.

4140 **Other** The remaining UD tags include punctuation (PUN) and symbols (SYM). Punc-
 4141 tuation is purely structural — e.g., commas, periods, colons — while symbols can carry
 4142 content of their own. Examples of symbols include dollar and percentage symbols, math-
 4143 ematical operators, emoticons, emojis, and internet addresses. A final catch-all tag is X,
 4144 which is used for words that cannot be assigned another part-of-speech category. The X
 4145 tag is also used in cases of **code switching** (between languages), described in § 8.5.

4146 **8.1.1.2 Other tagsets**

4147 Prior to the Universal Dependency treebank, part-of-speech tagging was performed us-
 4148 ing language-specific tagsets. The dominant tagset for English was designed as part of
 4149 the **Penn Treebank** (PTB), and it includes 45 tags — more than three times as many as
 4150 the UD tagset. This granularity is reflected in distinctions between singular and plural
 4151 nouns, verb tenses and aspects, possessive and non-possessive pronouns, comparative
 4152 and superlative adjectives and adverbs (e.g., *faster, fastest*), and so on. The Brown corpus
 4153 includes a tagset that is even more detailed, with 87 tags Francis (1964), including special
 4154 tags for individual auxiliary verbs such as *be, do, and have*.

4155 Different languages make different distinctions, and so the PTB and Brown tagsets are
 4156 not appropriate for a language such as Chinese, which does not mark the verb tense (Xia,
 4157 2000); nor for Spanish, which marks every combination of person and number in the
 4158 verb ending; nor for German, which marks the case of each noun phrase. Each of these
 4159 languages requires more detail than English in some areas of the tagset, and less in other
 4160 areas. The strategy of the Universal Dependencies corpus is to design a coarse-grained
 4161 tagset to be used across all languages, and then to additionally annotate language-specific
 4162 **morphosyntactic attributes**, such as number, tense, and case. The attribute tagging task
 4163 is described in more detail in § 8.2.

4164 Social media such as Twitter have been shown to require tagsets of their own (Gimpel
 4165 et al., 2011). Such corpora contain some tokens that are not equivalent to anything en-
 4166 countered in a typical written corpus: e.g., emoticons, URLs, and hashtags. Social media
 4167 also includes dialectal words like *gonna* ('going to', e.g. *We gonna be fine*) and *Ima* ('I'm
 4168 going to', e.g., *Ima tell you one more time*), which can be analyzed either as non-standard
 4169 orthography (making tokenization impossible), or as lexical items in their own right. In
 4170 either case, it is clear that existing tags like NOUN and VERB cannot handle cases like *Ima*,
 4171 which combine aspects of the noun and verb. Gimpel et al. (2011) therefore propose a new
 4172 set of tags to deal with these cases.

4173 **8.1.2 Accurate part-of-speech tagging**

4174 Part-of-speech tagging is the problem of selecting the correct tag for each word in a sen-
 4175 tence. Success is typically measured by accuracy on an annotated test set, which is simply
 4176 the fraction of tokens that were tagged correctly.

4177 8.1.2.1 Baselines

4178 A simple baseline for part-of-speech tagging is to choose the most common tag for each
4179 word. For example, in the Universal Dependencies treebank, the word *talk* appears 96
4180 times, and 85 of those times it is labeled as a VERB: therefore, this baseline will always
4181 predict VERB for this word. For words that do not appear in the training corpus, the base-
4182 line simply guesses the most common tag overall, which is NOUN. In the Penn Treebank,
4183 this simple baseline obtains accuracy above 92%. A more rigorous evaluation is the accu-
4184 racy on **out-of-vocabulary words**, which are not seen in the training data. Tagging these
4185 words correctly requires attention to the context and the word's internal structure.

4186 8.1.2.2 Contemporary approaches

4187 Conditional random fields and structured perceptron perform at or near the state-of-the-
4188 art for part-of-speech tagging in English. For example, (Collins, 2002) achieved 97.1%
4189 accuracy on the Penn Treebank, using a structured perceptron with the following base
4190 features (originally introduced by Ratnaparkhi (1996)):

- 4191 • current word, w_m
- 4192 • previous words, w_{m-1}, w_{m-2}
- 4193 • next words, w_{m+1}, w_{m+2}
- 4194 • previous tag, y_{m-1}
- 4195 • previous two tags, (y_{m-1}, y_{m-2})
- 4196 • for rare words:
 - 4197 – first k characters, up to $k = 4$
 - 4198 – last k characters, up to $k = 4$
 - 4199 – whether w_m contains a number, uppercase character, or hyphen.

4200 Similar results for the PTB data have been achieved using conditional random fields (CRFs;
4201 Toutanova et al., 2003).

4202 More recent work has demonstrated the power of neural sequence models, such as the
4203 **long short-term memory (LSTM)** (§ 7.6). Plank et al. (2016) apply a CRF and a bidirec-
4204 tional LSTM to twenty-two languages in the UD corpus, achieving an average accuracy
4205 of 94.3% for the CRF, and 96.5% with the bi-LSTM. Their neural model employs three
4206 types of embeddings: fine-tuned word embeddings, which are updated during training;
4207 pre-trained word embeddings, which are never updated, but which help to tag out-of-
4208 vocabulary words; and character-based embeddings. The character-based embeddings
4209 are computed by running an LSTM on the individual characters in each word, thereby
4210 capturing common orthographic patterns such as prefixes, suffixes, and capitalization.
4211 Extensive evaluations show that these additional embeddings are crucial to their model's
4212 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.

4213 8.2 Morphosyntactic Attributes

4214 There is considerably more to say about a word than whether it is a noun or a verb: in En-
 4215 glish, verbs are distinguish by features such tense and aspect, nouns by number, adjectives
 4216 by degree, and so on. These features are language-specific: other languages distinguish
 4217 other features, such as **case** (the role of the noun with respect to the action of the sen-
 4218 tence, which is marked in languages such as Latin and German⁵) and **evidentiality** (the
 4219 source of information for the speaker’s statement, which is marked in languages such as
 4220 Turkish). In the UD corpora, these attributes are annotated as feature-value pairs for each
 4221 token.⁶

4222 An example is shown in Figure 8.1. The determiner *the* is marked with two attributes:
 4223 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).

4262 stracts.

4263 A standard approach to tagging named entity spans is to use discriminative sequence
 4264 labeling methods such as conditional random fields. However, the named entity recogni-
 4265 tion (NER) task would seem to be fundamentally different from sequence labeling tasks
 4266 like part-of-speech tagging: rather than tagging each token, the goal is to recover *spans*
 4267 of tokens, such as *The United States Army*.

4268 This is accomplished by the **BIO notation**, shown in Figure 8.2. Each token at the
 4269 beginning of a name span is labeled with a B- prefix; each token within a name span is la-
 4270 beled with an I- prefix. These prefixes are followed by a tag for the entity type, e.g. B-LOC
 4271 for the beginning of a location, and I-PROTEIN for the inside of a protein name. Tokens
 4272 that are not parts of name spans are labeled as O. From this representation, the entity
 4273 name spans can be recovered unambiguously. This tagging scheme is also advantageous
 4274 for learning: tokens at the beginning of name spans may have different properties than
 4275 tokens within the name, and the learner can exploit this. This insight can be taken even
 4276 further, with special labels for the last tokens of a name span, and for unique tokens in
 4277 name spans, such as *Atlanta* in the example in Figure 8.2. This is called BILOU notation,
 4278 and it can yield improvements in supervised named entity recognition (Ratinov and Roth,
 4279 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*)

4280 Features can also be obtained from a **gazetteer**, which is a list of known entity names. For
 4281 example, the U.S. Social Security Administration provides a list of tens of thousands of

- (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)

4282 given names — more than could be observed in any annotated corpus. Tokens or spans
 4283 that match an entry in a gazetteer can receive special features; this provides a way to
 4284 incorporate hand-crafted resources such as name lists in a learning-driven framework.

4285 **Neural sequence labeling for NER** Current research has emphasized neural sequence
 4286 labeling, using similar LSTM models to those employed in part-of-speech tagging (Ham-
 4287 merton, 2003; Huang et al., 2015; Lample et al., 2016). The bidirectional LSTM-CRF (Fig-
 4288 ure 7.4 in § 7.6) does particularly well on this task, due to its ability to model tag-to-tag
 4289 dependencies. However, Strubell et al. (2017) show that **convolutional neural networks**
 4290 can be equally accurate, with significant improvement in speed due to the efficiency of
 4291 implementing ConvNets on **graphics processing units (GPUs)**. The key innovation in
 4292 this work was the use of **dilated convolution**, which is described in more detail in § 3.4.

4293 8.4 Tokenization

4294 A basic problem for text analysis, first discussed in § 4.3.1, is to break the text into a se-
 4295 quence of discrete tokens. For alphabetic languages such as English, deterministic scripts
 4296 usually suffice to achieve accurate tokenization. However, in logographic writing systems
 4297 such as Chinese script, words are typically composed of a small number of characters,
 4298 without intervening whitespace. The tokenization must be determined by the reader, with
 4299 the potential for occasional ambiguity, as shown in Figure 8.3. One approach is to match
 4300 character sequences against a known dictionary (e.g., Sproat et al., 1996), using additional
 4301 statistical information about word frequency. However, no dictionary is completely com-
 4302 prehensive, and dictionary-based approaches can struggle with such out-of-vocabulary
 4303 words.

4304 Chinese tokenization has therefore been approached as a supervised sequence label-
 4305 ing problem. Xue et al. (2003) train a logistic regression classifier to make independent
 4306 segmentation decisions while moving a sliding window across the document. A set of
 4307 rules is then used to convert these individual classification decisions into an overall tok-
 4308 enization of the input. However, these individual decisions may be globally suboptimal,
 4309 motivating a structure prediction approach. Peng et al. (2004) train a conditional random

4310 field to predict labels of START or NONSTART on each character. More recent work has
 4311 employed neural network architectures. For example, Chen et al. (2015) use an LSTM-
 4312 CRF architecture, as described in § 7.6: they construct a trellis, in which each tag is scored
 4313 according to the hidden state of an LSTM, and tag-tag transitions are scored according
 4314 to learned transition weights. The best-scoring segmentation is then computed by the
 4315 Viterbi algorithm.

4316 8.5 Code switching

4317 Multilingual speakers and writers do not restrict themselves to a single language. **Code**
4318 **switching** is the phenomenon of switching between languages in speech and text (Auer,
4319 2013; Poplack, 1980). Written code switching has become more common in online social
4320 media, as in the following extract from the website of Canadian President Justin Trudeau:⁷

- 4321 (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*

4323 Accurately analyzing such texts requires first determining which languages are being
4324 used. Furthermore, quantitative analysis of code switching can provide insights on the
4325 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)

4341 8.6 Dialogue acts

4342 The sequence labeling problems that we have discussed so far have been over sequences
 4343 of word tokens or characters (in the case of tokenization). However, sequence labeling
 4344 can also be performed over higher-level units, such as **utterances**. **Dialogue acts** are la-
 4345 bels over utterances in a dialogue, corresponding roughly to the speaker’s intention —
 4346 the utterance’s **illocutionary force** (Austin, 1962). For example, an utterance may state a
 4347 proposition (*it is not down on any map*), pose a question (*shall we keep chasing this murderous*
 4348 *fish?*), or provide a response (*aye aye!*). Stolcke et al. (2000) describe how a set of 42 dia-
 4349 logue acts were annotated for the 1,155 conversations in the Switchboard corpus (Godfrey
 4350 et al., 1992).⁸

4351 An example is shown in Figure 8.4. The annotation is performed over UTTERANCES,
 4352 with the possibility of multiple utterances per **conversational turn** (in cases such as inter-
 4353 ruptions, an utterance may split over multiple turns). Some utterances are clauses (e.g., *So*
 4354 *do you go to college right now?*), while others are single words (e.g., *yeah*). Stolcke et al. (2000)
 4355 report that hidden Markov models (HMMs) achieve 96% accuracy on supervised utter-
 4356 ance segmentation. The labels themselves reflect the conversational goals of the speaker:
 4357 the utterance *yeah* functions as an answer in response to the question *you’re a senior now*,
 4358 but in the final line of the excerpt, it is a **backchannel** (demonstrating comprehension).

4359 For task of dialogue act labeling, Stolcke et al. (2000) apply a hidden Markov model.
 4360 The probability $p(w_m | y_m)$ must generate the entire sequence of words in the utterance,
 4361 and it is modeled as a trigram language model (§ 6.1). Stolcke et al. (2000) also account
 4362 for acoustic features, which capture the **prosody** of each utterance — for example, tonal
 4363 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.

4364 as questions and answers. These features are handled with an additional emission distri-
 4365 bution, $p(a_m | y_m)$, which is modeled with a probabilistic decision tree (Murphy, 2012).
 4366 While acoustic features yield small improvements overall, they play an important role in
 4367 distinguish questions from statements, and agreements from backchannels.

4368 Recurrent neural architectures for dialogue act labeling have been proposed by Kalch-
 4369 brenner and Blunsom (2013) and Ji et al. (2016), with strong empirical results. Both models
 4370 are recurrent at the utterance level, so that each complete utterance updates a hidden state.
 4371 The recurrent-convolutional network of Kalchbrenner and Blunsom (2013) uses convolu-
 4372 tion to obtain a representation of each individual utterance, while Ji et al. (2016) use a
 4373 second level of recurrence, over individual words. This enables their method to also func-
 4374 tion as a language model, giving probabilities over sequences of words in a document.

4375 Exercises

4376 1. Using the Universal Dependencies part-of-speech tags, annotate the following sen-
 4377 tences. You may examine the UD tagging guidelines. Tokenization is shown with
 4378 whitespace. Don't forget about punctuation.

4379 (8.40) I try all things , I achieve what I can .

4380 (8.41) It was that accursed white whale that razed me .

4381 (8.42) Better to sleep with a sober cannibal , than a drunk Christian .

4382 (8.43) Be it what it will , I 'll go to it laughing .

4383 2. Select three short sentences from a recent news article, and annotate them for UD
 4384 part-of-speech tags. Ask a friend to annotate the same three sentences without look-
 4385 ing at your annotations. Compute the rate of agreement, using the Kappa metric
 4386 defined in § 4.5.2.1. Then work together to resolve any disagreements.

4387 3. Choose one of the following morphosyntactic attributes: MOOD, TENSE, VOICE. Re-
 4388 search the definition of this attribute on the universal dependencies website, <http://universaldependencies.org/u/feat/index.html>. Returning to the ex-
 4389 amples in the first exercise, annotate all verbs for your chosen attribute. It may be
 4390 helpful to consult examples from an English-language universal dependencies cor-
 4391 pus, available at https://github.com/UniversalDependencies/UD_English-EWT/tree/master.

4394 4. Download a dataset annotated for universal dependencies, such as the English Tree-
 4395 bank at https://github.com/UniversalDependencies/UD_English-EWT/tree/master. This corpus is already segmented into training, development, and
 4396 test data.
 4397

- 4398 a) First, train a logistic regression or SVM classifier using character suffixes: char-
 4399 acter n-grams up to length 4. Compute the recall, precision, and *F*-MEASURE
 4400 on the development data.
- 4401 b) Next, augment your classifier using the same character suffixes of the preced-
 4402 ing and succeeding tokens. Again, evaluate your classifier on heldout data.
- 4403 c) Optionally, train a Viterbi-based sequence labeling model, using a toolkit such
 4404 as CRFSuite (<http://www.chokkan.org/software/crfsuite/>) or your
 4405 own Viterbi implementation. This is more likely to be helpful for attributes
 4406 in which agreement is required between adjacent words. For example, many
 4407 Romance languages require gender and number agreement for determiners,
 4408 nouns, and adjectives.
- 4409 5. Provide BIO-style annotation of the named entities (person, place, organization,
 4410 date, or product) in the following expressions:
- 4411 (8.44) The third mate was Flask, a native of Tisbury, in Martha's Vineyard.
- 4412 (8.45) Its official Nintendo announced today that they Will release the Nintendo
 4413 3DS in north America march 27 (Ritter et al., 2011).
- 4414 (8.46) Jessica Reif, a media analyst at Merrill Lynch & Co., said, "If they can get
 4415 up and running with exclusive programming within six months, it doesn't
 4416 set the venture back that far."⁹
- 4417 6. Run the examples above through the online version of a named entity recogni-
 4418 tion tagger, such as the Allen NLP system here: <http://demo.allennlp.org/named->
 4419 entity-recognition. Do the predicted tags match your annotations?
- 4420 7. Build a whitespace tokenizer for English:
- 4421 a) Using the NLTK library, download the complete text to the novel *Alice in Won-*
 4422 *derland* (Carroll, 1865). Hold out the final 1000 words as a test set.
- 4423 b) Label each alphanumeric character as a segmentation point, $y_m = 1$ if m is
 4424 the final character of a token. Label every other character as $y_m = 0$. Then
 4425 concatenate all the tokens in the training and test sets. Make sure that the num-
 4426 ber of labels $\{y_m\}_{m=1}^M$ is identical to the number of characters $\{c_m\}_{m=1}^M$ in your
 4427 concatenated datasets.
- 4428 c) Train a logistic regression classifier to predict y_m , using the surrounding char-
 4429 acters $c_{m-5:m+5}$ as features. After training the classifier, run it on the test set,
 4430 using the predicted segmentation points to re-tokenize the text.

⁹From the Message Understanding Conference (MUC-7) dataset (Chinchor and Robinson, 1997).

- 4431 d) Compute the per-character segmentation accuracy on the test set. You should
4432 be able to get at least 88% accuracy.
4433 e) Print out a sample of segmented text from the test set, e.g.

4434 Thereareno mice in the air , I ' m afraid , but y oumight cat
4435 chabat , and that ' s very like a mouse , youknow . But
4436 docatseat bats , I wonder ?'

- 4437 8. Perform the following extensions to your tokenizer in the previous problem.

- 4438 a) Train a conditional random field sequence labeler, by incorporating the tag
4439 bigrams (y_{m-1}, y_m) as additional features. You may use a structured predic-
4440 tion library such as CRFSuite, or you may want to implement Viterbi yourself.
4441 Compare the accuracy with your classification-based approach.
- 4442 b) Compute the token-level performance: treating the original tokenization as
4443 ground truth, compute the number of true positives (tokens that are in both
4444 the ground truth and predicted tokenization), false positives (tokens that are in
4445 the predicted tokenization but not the ground truth), and false negatives (to-
4446 kens that are in the ground truth but not the predicted tokenization). Compute
4447 the F-measure.
4448 Hint: to match predicted and ground truth tokens, add “anchors” for the start
4449 character of each token. The number of true positives is then the size of the
4450 intersection of the sets of predicted and ground truth tokens.
- 4451 c) Apply the same methodology in a more practical setting: tokenization of Chi-
4452 nese, which is written without whitespace. You can find annotated datasets at
4453 <http://alias-i.com/lingpipe/demos/tutorial/chineseTokens/read-me.html>.

4455

Chapter 9

4456

Formal language theory

4457 We have now seen methods for learning to label individual words, vectors of word counts,
4458 and sequences of words; we will soon proceed to more complex structural transfor-
4459 mations. Most of these techniques could apply to counts or sequences from any discrete vo-
4460 cabulary; there is nothing fundamentally linguistic about, say, a hidden Markov model.
4461 This raises a basic question that this text has not yet considered: what is a language?

4462 This chapter will take the perspective of **formal language theory**, in which a language
4463 is defined as a set of **strings**, each of which is a sequence of elements from a finite alphabet.
4464 For interesting languages, there are an infinite number of strings that are in the language,
4465 and an infinite number of strings that are not. For example:

- 4466 • the set of all even-length sequences from the alphabet $\{a, b\}$, e.g., $\{\emptyset, aa, ab, ba, bb, aaaa, aaab, \dots\}$;
- 4467 • the set of all sequences from the alphabet $\{a, b\}$ that contain *aaa* as a substring, e.g.,
4468 $\{aaa, aaaa, baaa, aaab, \dots\}$;
- 4469 • the set of all sequences of English words (drawn from a finite dictionary) that con-
4470 tain at least one verb (a finite subset of the dictionary);
- 4471 • the `python` programming language.

4472 Formal language theory defines classes of languages and their computational prop-
4473 erties. Of particular interest is the computational complexity of solving the **membership**
4474 **problem** — determining whether a string is in a language. The chapter will focus on
4475 three classes of formal languages: regular, context-free, and “mildly” context-sensitive
4476 languages.

4477 A key insight of 20th century linguistics is that formal language theory can be usefully
4478 applied to natural languages such as English, by designing formal languages that cap-
4479 ture as many properties of the natural language as possible. For many such formalisms, a
4480 useful linguistic analysis comes as a byproduct of solving the membership problem. The

4481 membership problem can be generalized to the problems of *scoring* strings for their ac-
 4482 ceptability (as in language modeling), and of **transducing** one string into another (as in
 4483 translation).

4484 9.1 Regular languages

4485 If you have written a **regular expression**, then you have defined a **regular language**: a
 4486 regular language is any language that can be defined by a regular expression. Formally, a
 4487 regular expression can include the following elements:

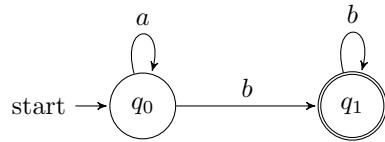
- 4488 • A **literal character** drawn from some finite alphabet Σ .
- 4489 • The **empty string** ϵ .
- 4490 • The concatenation of two regular expressions RS , where R and S are both regular
 4491 expressions. The resulting expression accepts any string that can be decomposed
 4492 $x = yz$, where y is accepted by R and z is accepted by S .
- 4493 • The alternation $R \mid S$, where R and S are both regular expressions. The resulting
 4494 expression accepts a string x if it is accepted by R or it is accepted by S .
- 4495 • The **Kleene star** R^* , which accepts any string x that can be decomposed into a se-
 4496 quence of strings which are all accepted by R .
- 4497 • Parenthesization ((R)), which is used to limit the scope of the concatenation, alterna-
 4498 tion, and Kleene star operators.

4499 Here are some example regular expressions:

- 4500 • The set of all even length strings on the alphabet $\{a, b\}$: $((aa)|(ab)|(ba)|(bb))^*$
- 4501 • The set of all sequences of the alphabet $\{a, b\}$ that contain aaa as a substring: $(a|b)^*aaa(a|b)^*$
- 4502 • The set of all sequences of English words that contain at least one verb: W^*VW^* ,
 4503 where W is an alternation between all words in the dictionary, and V is an alterna-
 4504 tion between all verbs ($V \subseteq W$).

4505 This list does not include a regular expression for the Python programming language,
 4506 because this language is not regular — there is no regular expression that can capture its
 4507 syntax. We will discuss why towards the end of this section.

4508 Regular languages are **closed** under union, intersection, and concatenation. This means
 4509 that if two languages L_1 and L_2 are regular, then so are the languages $L_1 \cup L_2$, $L_1 \cap L_2$,
 4510 and the language of strings that can be decomposed as $s = tu$, with $s \in L_1$ and $t \in L_2$.
 4511 Regular languages are also closed under negation: if L is regular, then so is the language
 4512 $\bar{L} = \{s \notin L\}$.

Figure 9.1: State diagram for the finite state acceptor M_1 .4513 **9.1.1 Finite state acceptors**

4514 A regular expression defines a regular language, but does not give an algorithm for de-
 4515 termining whether a string is in the language that it defines. **Finite state automata** are
 4516 theoretical models of computation on regular languages, which involve transitions be-
 4517 tween a finite number of states. The most basic type of finite state automaton is the **finite**
 4518 **state acceptor (FSA)**, which describes the computation involved in testing if a string is
 4519 a member of a language. Formally, a finite state acceptor is a tuple $M = (Q, \Sigma, q_0, F, \delta)$,
 4520 consisting of:

- 4521 • a finite alphabet Σ of input symbols;
- 4522 • a finite set of states $Q = \{q_0, q_1, \dots, q_n\}$;
- 4523 • a start state $q_0 \in Q$;
- 4524 • a set of final states $F \subseteq Q$;
- 4525 • a transition function $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$. The transition function maps from a
 4526 state and an input symbol (or empty string ϵ) to a *set* of possible resulting states.

4527 A **path** in M is a sequence of transitions, $\pi = t_1, t_2, \dots, t_N$, where each t_i traverses an
 4528 arc in the transition function δ . The finite state acceptor M accepts a string ω if there is
 4529 an accepting path, in which the initial transition t_1 begins at the start state q_0 , the final
 4530 transition t_N terminates in a final state in Q , and the entire input ω is consumed.

4531 **9.1.1.1 Example**

Consider the following FSA, M_1 .

$$\Sigma = \{a, b\} \quad [9.1]$$

$$Q = \{q_0, q_1\} \quad [9.2]$$

$$F = \{q_1\} \quad [9.3]$$

$$\delta = \{(q_0, a) \rightarrow q_0, (q_0, b) \rightarrow q_1, (q_1, b) \rightarrow q_1\}. \quad [9.4]$$

4532 This FSA defines a language over an alphabet of two symbols, a and b . The transition
 4533 function δ is written as a set of arcs: $(q_0, a) \rightarrow q_0$ says that if the machine is in state

4534 q_0 and reads symbol a , it stays in q_0 . Figure 9.1 provides a graphical representation of
 4535 M_1 . Because each pair of initial state and symbol has at most one resulting state, M_1 is
 4536 **deterministic**: each string ω induces at most one accepting path. Note that there are no
 4537 transitions for the symbol a in state q_1 ; if a is encountered in q_1 , then the acceptor is stuck,
 4538 and the input string is rejected.

4539 What strings does M_1 accept? The start state is q_0 , and we have to get to q_1 , since this
 4540 is the only final state. Any number of a symbols can be consumed in q_0 , but a b symbol is
 4541 required to transition to q_1 . Once there, any number of b symbols can be consumed, but
 4542 an a symbol cannot. So the regular expression corresponding to the language defined by
 4543 M_1 is a^*bb^* .

4544 9.1.1.2 Computational properties of finite state acceptors

4545 The key computational question for finite state acceptors is: how fast can we determine
 4546 whether a string is accepted? For deterministic FSAs, this computation can be performed
 4547 by Dijkstra's algorithm, with time complexity $\mathcal{O}(V \log V + E)$, where V is the number of
 4548 vertices in the FSA, and E is the number of edges (Cormen et al., 2009). Non-deterministic
 4549 FSAs (NFSAs) can include multiple transitions from a given symbol and state. Any NSFA
 4550 can be converted into a deterministic FSA, but the resulting automaton may have a num-
 4551 ber of states that is exponential in the number of size of the original NFSFA (Mohri et al.,
 4552 2002).

4553 9.1.2 Morphology as a regular language

4554 Many words have internal structure, such as prefixes and suffixes that shape their mean-
 4555 ing. The study of word-internal structure is the domain of **morphology**, of which there
 4556 are two main types:

- 4557 • **Derivational morphology** describes the use of affixes to convert a word from one
 4558 grammatical category to another (e.g., from the noun *grace* to the adjective *graceful*),
 4559 or to change the meaning of the word (e.g., from *grace* to *disgrace*).
- 4560 • **Inflectional morphology** describes the addition of details such as gender, number,
 4561 person, and tense (e.g., the *-ed* suffix for past tense in English).

4562 Morphology is a rich topic in linguistics, deserving of a course in its own right.¹ The
 4563 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).

4564 current section deals with derivational morphology; inflectional morphology is discussed
4565 in § 9.1.4.3.

4566 Suppose that we want to write a program that accepts only those words that are con-
4567 structed in accordance with the rules of English derivational morphology:

- 4568 (9.1) grace, graceful, gracefully, *gracelyful
4569 (9.2) disgrace, *ungrace, disgraceful, disgracefully
4570 (9.3) allure, *allureful, alluring, alluringly
4571 (9.4) fairness, unfair, *disfair, fairly

4572 (Recall that the asterisk indicates that a linguistic example is judged unacceptable by flu-
4573 ent speakers of a language.) These examples cover only a tiny corner of English deriva-
4574 tional morphology, but a number of things stand out. The suffix *-ful* converts the nouns
4575 *grace* and *disgrace* into adjectives, and the suffix *-ly* converts adjectives into adverbs. These
4576 suffixes must be applied in the correct order, as shown by the unacceptability of **grace-*
4577 *lyful*. The *-ful* suffix works for only some words, as shown by the use of *alluring* as the
4578 adjectival form of *allure*. Other changes are made with prefixes, such as the derivation
4579 of *disgrace* from *grace*, which roughly corresponds to a negation; however, *fair* is negated
4580 with the *un-* prefix instead. Finally, while the first three examples suggest that the direc-
4581 tion of derivation is noun → adjective → adverb, the example of *fair* suggests that the
4582 adjective can also be the base form, with the *-ness* suffix performing the conversion to a
4583 noun.

4584 Can we build a computer program that accepts only well-formed English words, and
4585 rejects all others? This might at first seem trivial to solve with a brute-force attack: simply
4586 make a dictionary of all valid English words. But such an approach fails to account for
4587 morphological **productivity** — the applicability of existing morphological rules to new
4588 words and names, such as *Trump* to *Trumpy* and *Trumpkin*, and *Clinton* to *Clintonian* and
4589 *Clintonite*. We need an approach that represents morphological rules explicitly, and for
4590 this we will try a finite state acceptor.

4591 The dictionary approach can be implemented as a finite state acceptor, with the vo-
4592 cabulary Σ equal to the vocabulary of English, and a transition from the start state to the
4593 accepting state for each word. But this would of course fail to generalize beyond the origi-
4594 nal vocabulary, and would not capture anything about the **morphotactic** rules that govern
4595 derivations from new words. The first step towards a more general approach is shown in
4596 Figure 9.2, which is the state diagram for a finite state acceptor in which the vocabulary
4597 consists of **morphemes**, which include **stems** (e.g., *grace*, *allure*) and **affixes** (e.g., *dis-*, *-ing*,
4598 *-ly*). This finite state acceptor consists of a set of paths leading away from the start state,
4599 with derivational affixes added along the path. Except for q_{neg} , the states on these paths
4600 are all final, so the FSA will accept *disgrace*, *disgraceful*, and *disgracefully*, but not *dis-*.

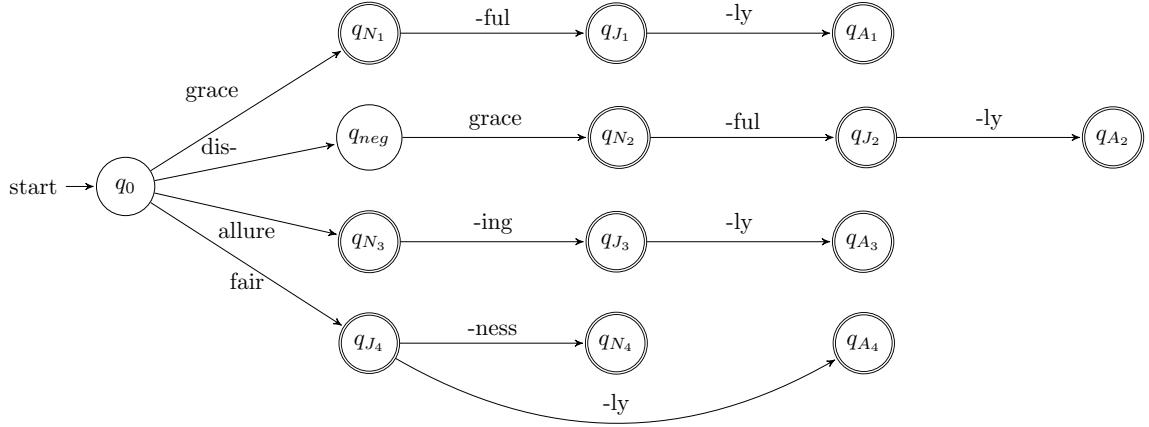


Figure 9.2: A finite state acceptor for a fragment of English derivational morphology. Each path represents possible derivations from a single root form.

4601 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
 4602 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
 4603 be extended: as new word stems are added to the vocabulary, their derived forms will be
 4604 accepted automatically. Of course, this FSA would still need to be extended considerably
 4605 to cover even this small fragment of English morphology. As shown by cases like *music*
 4606 → *musical*, *athlete* → *athletic*, English includes several classes of nouns, each with its own
 4607 rules for derivation.
 4608

4610 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 can be handled by **finite state transducers**, which are finite state automata that take inputs and produce outputs.

4617 9.1.3 Weighted finite state acceptors

4618 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
 4619 English words; but if forced to choose, you probably find *a fishful stew* or *a musicky trib-*
 4620 *ute* preferable to *behaving disgracelyful*. Rather than asking whether a word is acceptable,
 4621 we might like to ask how acceptable it is. Aronoff (1976, page 36) puts it another way:
 4622

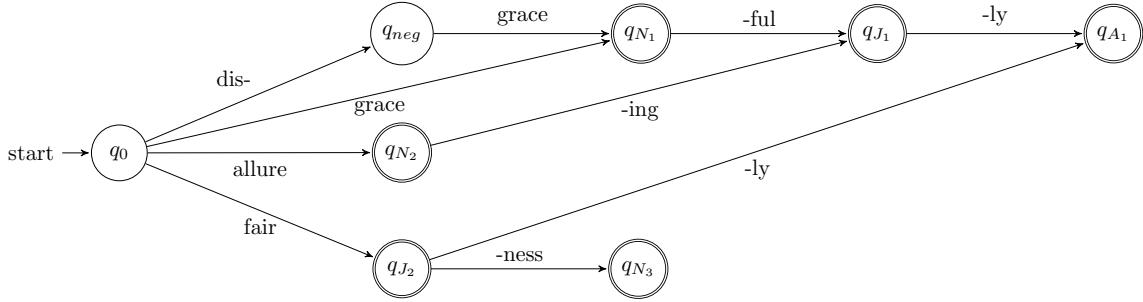


Figure 9.3: Minimization of the finite state acceptor shown in Figure 9.2.

4623 “Though many things are possible in morphology, some are more possible than others.”
 4624 But finite state acceptors give no way to express preferences among technically valid
 4625 choices.

4626 **Weighted finite state acceptors (WFSAs)** are generalizations of FSAs, in which each
 4627 accepting path is assigned a score, computed from the transitions, the initial state, and the
 4628 final state. Formally, a weighted finite state acceptor $M = (Q, \Sigma, \lambda, \rho, \delta)$ consists of:

- 4629 • a finite set of states $Q = \{q_0, q_1, \dots, q_n\}$;
- 4630 • a finite alphabet Σ of input symbols;
- 4631 • an initial weight function, $\lambda : Q \rightarrow \mathbb{R}$;
- 4632 • a final weight function $\rho : Q \rightarrow \mathbb{R}$;
- 4633 • a transition function $\delta : Q \times \Sigma \times Q \rightarrow \mathbb{R}$.

4634 WFSAs depart from the FSA formalism in three ways: every state can be an initial
 4635 state, with score $\lambda(q)$; every state can be an accepting state, with score $\rho(q)$; transitions are
 4636 possible between any pair of states on any input, with a score $\delta(q_i, \omega, q_j)$. Nonetheless,
 4637 FSAs can be viewed as a special case: for any FSA M we can build an equivalent WFSA
 4638 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
 4639 transitions $\{(q_1, \omega) \rightarrow q_2\}$ that are not permitted by the transition function of M .

4640 The total score for any path $\pi = t_1, t_2, \dots, t_N$ is equal to the sum of these scores,

$$d(\pi) = \lambda(\text{from-state}(t_1)) + \sum_n^N \delta(t_n) + \rho(\text{to-state}(t_N)). \quad [9.5]$$

4641 A **shortest-path algorithm** is used to find the minimum-cost path through a WFSA for
 4642 string ω , with time complexity $\mathcal{O}(E + V \log V)$, where E is the number of edges and V is
 4643 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

4644 **9.1.3.1 N-gram language models as WFSAs**

4645 In **n-gram language models** (see § 6.1), the probability of a sequence of tokens w_1, w_2, \dots, w_M
 4646 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:

$$\begin{aligned} \delta(q_i, \omega, q_j) &= \begin{cases} \log \Pr(w_m = j | w_{m-1} = i), & \omega = j \\ -\infty, & \omega \neq j \end{cases} \\ \lambda(q_i) &= \log \Pr(w_1 = i | w_0 = \square) \\ \rho(q_i) &= \log \Pr(w_{M+1} = \blacksquare | w_M = i). \end{aligned}$$

4647 The transition function is designed to ensure that the context is recorded accurately:
 4648 we can move to state j on input ω only if $\omega = j$; otherwise, transitioning to state j is
 4649 forbidden by the weight of $-\infty$. The initial weight function $\lambda(q_i)$ is the log probability of
 4650 receiving i as the first token, and the final weight function $\rho(q_i)$ is the log probability of
 4651 receiving an “end-of-string” token after observing $w_M = i$.

4652 **9.1.3.2 *Semiring weighted finite state acceptors**

4653 The n -gram language model WFSA is deterministic: each input has exactly one accepting
 4654 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.

4655 may have multiple accepting paths. In some applications, the score for the input is ag-
 4656 gregated across all such paths. Such aggregate scores can be computed by generalizing
 4657 WFSAs with **semiring notation**, first introduced in § 7.7.3.

4658 Let $d(\pi)$ represent the total score for path $\pi = t_1, t_2, \dots, t_N$, which is computed as,

$$d(\pi) = \lambda(\text{from-state}(t_1)) \otimes \delta(t_1) \otimes \delta(t_2) \otimes \dots \otimes \delta(t_N) \otimes \rho(\text{to-state}(t_N)). \quad [9.8]$$

4659 This is a generalization of Equation 9.5 to semiring notation, using the semiring multipli-
 4660 cation operator \otimes in place of addition.

4661 Now let $s(\omega)$ represent the total score for all paths $\Pi(\omega)$ that consume input ω ,

$$s(\omega) = \bigoplus_{\pi \in \Pi(\omega)} d(\pi). \quad [9.9]$$

4662 Here, semiring addition (\oplus) is used to combine the scores of multiple paths.

4663 The generalization to semirings covers a number of useful special cases. In the log-
 4664 probability semiring, multiplication is defined as $\log p(x) \otimes \log p(y) = \log p(x) + \log p(y)$,
 4665 and addition is defined as $\log p(x) \oplus \log p(y) = \log(p(x) + p(y))$. Thus, $s(\omega)$ represents
 4666 the log-probability of accepting input ω , marginalizing over all paths $\pi \in \Pi(\omega)$. In the
 4667 **boolean semiring**, the \otimes operator is logical conjunction, and the \oplus operator is logical
 4668 disjunction. This reduces to the special case of unweighted finite state acceptors, where
 4669 the score $s(\omega)$ is a boolean indicating whether there exists any accepting path for ω . In
 4670 the **tropical semiring**, the \oplus operator is a maximum, so the resulting score is the score of
 4671 the best-scoring path through the WFSAs. The OpenFST toolkit uses semirings and poly-
 4672 morphism to implement general algorithms for weighted finite state automata (Allauzen
 4673 et al., 2007).

4674 9.1.3.3 *Interpolated n -gram language models

4675 Recall from § 6.2.3 that an **interpolated n -gram language model** combines the probabili-
 4676 ties from multiple n -gram models. For example, an interpolated bigram language model
 4677 computes the probability,

$$\hat{p}(w_m | w_{m-1}) = \lambda_1 p_1(w_m) + \lambda_2 p_2(w_m | w_{m-1}), \quad [9.10]$$

4678 with \hat{p} indicating the interpolated probability, p_2 indicating the bigram probability, and
 4679 p_1 indicating the unigram probability. Setting $\lambda_2 = (1 - \lambda_1)$ ensures that the probabilities
 4680 sum to one.

4681 Interpolated bigram language models can be implemented using a non-deterministic
 4682 WFSAs (Knight and May, 2009). The basic idea is shown in Figure 9.4. In an interpolated
 4683 bigram language model, there is one state for each element in the vocabulary — in this

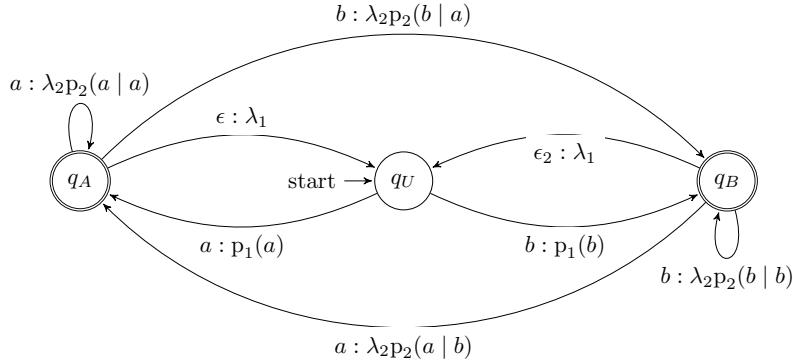


Figure 9.4: WFSA implementing an interpolated bigram/unigram language model, on the alphabet $\Sigma = \{a, b\}$. For simplicity, the WFSA is constrained to force the first token to be generated from the unigram model, and does not model the emission of the end-of-sequence token.

4684 case, the states q_A and q_B — which capture the contextual conditioning in the bigram
 4685 probabilities. To model unigram probabilities, there is an additional state q_U , which “for-
 4686 gets” the context. Transitions out of q_U involve unigram probabilities, $p_1(a)$ and $p_2(b)$;
 4687 transitions into q_U emit the empty symbol ϵ , and have probability λ_1 , reflecting the inter-
 4688 polation weight for the unigram model. The interpolation weight for the bigram model is
 4689 included in the weight of the transition $q_A \rightarrow q_B$.

4690 The epsilon transitions into q_U make this WFSA non-deterministic. Consider the score
 4691 for the sequence (a, b, b) . The initial state is q_U , so the symbol a is generated with score
 4692 $p_1(a)$ ³ Next, we can generate b from the unigram model by taking the transition $q_A \rightarrow q_B$,
 4693 with score $\lambda_2 p_2(b | a)$. Alternatively, we can take a transition back to q_U with score λ_1 ,
 4694 and then emit b from the unigram model with score $p_1(b)$. To generate the final b token,
 4695 we face the same choice: emit it directly from the self-transition to q_B , or transition to q_U
 4696 first.

The total score for the sequence (a, b, b) is the semiring sum over all accepting paths,

$$\begin{aligned}
 s(a, b, b) &= (p_1(a) \otimes \lambda_2 p_2(b | a) \otimes \lambda_2 p_2(b | b)) \\
 &\oplus (p_1(a) \otimes \lambda_1 \otimes p_1(b) \otimes \lambda_2 p_2(b | b)) \\
 &\oplus (p_1(a) \otimes \lambda_2 p_2(b | a) \otimes p_1(b) \otimes p_1(b)) \\
 &\oplus (p_1(a) \otimes \lambda_1 \otimes p_1(b) \otimes p_1(b) \otimes p_1(b)). \tag{[9.11]}
 \end{aligned}$$

4697 Each line in Equation 9.11 represents the probability of a specific path through the WFSA.
 4698 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.

4699 transition weight, which are themselves probabilities. The \oplus operator is addition, so that
 4700 the total score is the sum of the scores (probabilities) for each path. This corresponds to
 4701 the probability under the interpolated bigram language model.

4702 9.1.4 Finite state transducers

4703 Finite state acceptors can determine whether a string is in a regular language, and weighted
 4704 finite state acceptors can compute a score for every string over a given alphabet. **Finite**
 4705 **state transducers** (FSTs) extend the formalism further, by adding an output symbol to each
 4706 transition. Formally, a finite state transducer is a tuple $T = (Q, \Sigma, \Omega, \lambda, \rho, \delta)$, with Ω repre-
 4707 senting an output vocabulary and the transition function $\delta : Q \times (\Sigma \cup \epsilon) \times (\Omega \cup \epsilon) \times Q \rightarrow \mathbb{R}$
 4708 mapping from states, input symbols, and output symbols to states. The remaining ele-
 4709 ments (Q, Σ, λ, ρ) are identical to their definition in weighted finite state acceptors (§ 9.1.3).
 4710 Thus, each path through the FST T transduces the input string into an output.

4711 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,

$$\delta(q, a, a, q) = \delta(q, b, b, q) = 0 \quad [9.12]$$

$$\delta(q, a, b, q) = \delta(q, b, a, q) = 1 \quad [9.13]$$

$$\delta(q, a, \epsilon, q) = \delta(q, b, \epsilon, q) = 1 \quad [9.14]$$

$$\delta(q, \epsilon, a, q) = \delta(q, \epsilon, b, q) = 1. \quad [9.15]$$

4712 The state diagram is shown in Figure 9.5.

4713 For a given string pair, there are multiple paths through the transducer: the best-
 4714 scoring path from *dessert* to *desert* involves a single deletion, for a total score of 1; the
 4715 worst-scoring path involves seven deletions and six additions, for a score of 13.

4716 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

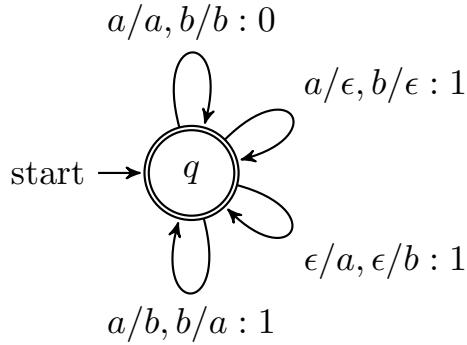


Figure 9.5: State diagram for the Levenshtein edit distance finite state transducer. The label $x/y : c$ indicates a cost of c for a transition with input x and output y .

by an unweighted finite state transducer. The first rule is:

$$-sses \rightarrow -ss \quad \text{e.g., } dresses \rightarrow dress \quad [9.16]$$

$$-ies \rightarrow -i \quad \text{e.g., } parties \rightarrow parti \quad [9.17]$$

$$-ss \rightarrow -ss \quad \text{e.g., } dress \rightarrow dress \quad [9.18]$$

$$-s \rightarrow \epsilon \quad \text{e.g., } cats \rightarrow cat \quad [9.19]$$

4717 The final two lines appear to conflict; they are meant to be interpreted as an instruction
 4718 to remove a terminal $-s$ unless it is part of an $-ss$ ending. A state diagram to handle just
 4719 these final two lines is shown in Figure 9.6. Make sure you understand how this finite
 4720 state transducer handles *cats*, *steps*, *bass*, and *basses*.

4721 9.1.4.3 Inflectional morphology

4722 In **inflectional morphology**, word **lemmas** are modified to add grammatical information
 4723 such as tense, number, and case. For example, many English nouns are pluralized by the
 4724 suffix $-s$, and many verbs are converted to past tense by the suffix $-ed$. English's inflectional
 4725 morphology is considerably simpler than many of the world's languages. For example,
 4726 Romance languages (derived from Latin) feature complex systems of verb suffixes which
 4727 must agree with the person and number of the verb, as shown in Table 9.1.

4728 The task of morphological analysis is to read a form like *canto*, and output an analysis
 4729 like CANTAR+VERB+PRESIND+1P+SING, where +PRESIND describes the tense as present
 4730 indicative, +1P indicates the first-person, and +SING indicates the singular number. The
 4731 task of morphological generation is the reverse, going from CANTAR+VERB+PRESIND+1P+SING
 4732 to *canto*. Finite state transducers are an attractive solution, because they can solve both
 4733 problems with a single model (Beesley and Karttunen, 2003). As an example, Figure 9.7
 4734 shows a fragment of a finite state transducer for Spanish inflectional morphology. The

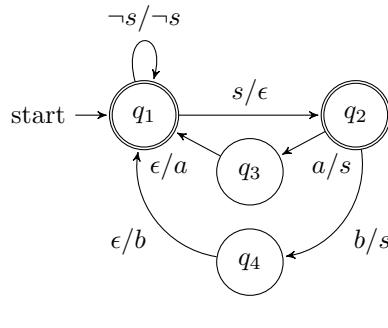


Figure 9.6: State diagram for final two lines of step 1a of the Porter stemming diagram. States q_3 and q_4 “remember” the observations a and b respectively; the ellipsis \dots represents additional states for each symbol in the input alphabet. The notation $\neg s / \neg s$ is not part of the FST formalism; 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.

4735 input vocabulary Σ corresponds to the set of letters used in Spanish spelling, and the out-
 4736 put vocabulary Ω corresponds to these same letters, plus the vocabulary of morphological
 4737 features (e.g., +SING, +VERB). In Figure 9.7, there are two paths that take *canto* as input,
 4738 corresponding to the verb and noun meanings; the choice between these paths could be
 4739 guided by a part-of-speech tagger. By **inversion**, the inputs and outputs for each trans-
 4740 sition are switched, resulting in a finite state generator, capable of producing the correct
 4741 **surface form** for any morphological analysis.

4742 Finite state morphological analyzers and other unweighted transducers can be de-
 4743 signed by hand. The designer’s goal is to avoid **overgeneration** — accepting strings or
 4744 making transductions that are not valid in the language — as well as **undergeneration**

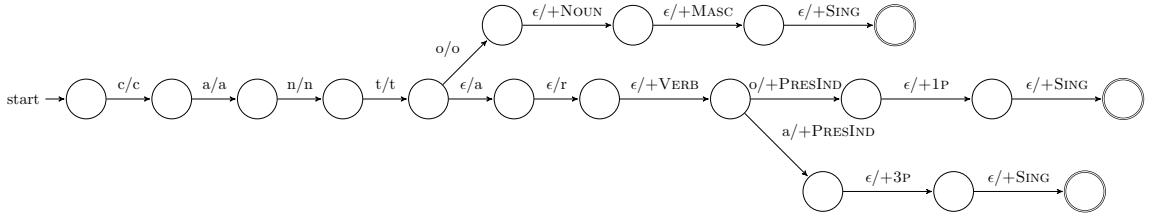


Figure 9.7: Fragment of a finite state transducer for Spanish morphology. There are two accepting paths for the input *canto*: *canto+NOUN+MASC+SING* (masculine singular noun, meaning a song), and *cantar+VERB+PRESIND+1P+SING* (I sing). There is also an accepting path for *canta*, with output *cantar+VERB+PRESIND+3P+SING* (he/she sings).

4745 — failing to accept strings or transductions that are valid. For example, a pluralization
 4746 transducer that does not accept *foot/feet* would undergenerate. Suppose we “fix” the trans-
 4747 ducer to accept this example, but as a side effect, it now accepts *boot/beet*; the transducer
 4748 would then be said to overgenerate. If a transducer accepts *foot/foots* but not *foot/feet*, then
 4749 it simultaneously overgenerates and undergenerates.

4750 9.1.4.4 Finite state composition

4751 Designing finite state transducers to capture the full range of morphological phenomena
 4752 in any real language is a huge task. Modularization is a classic computer science approach
 4753 for this situation: decompose a large and unwieldy problem into a set of subproblems,
 4754 each of which will hopefully have a concise solution. Finite state automata can be mod-
 4755 ularized through **composition**: feeding the output of one transducer T_1 as the input to
 4756 another transducer T_2 , written $T_2 \circ T_1$. Formally, if there exists some y such that $(x, y) \in T_1$
 4757 (meaning that T_1 produces output y on input x), and $(y, z) \in T_2$, then $(x, z) \in (T_2 \circ T_1)$.
 4758 Because finite state transducers are closed under composition, there is guaranteed to be
 4759 a single finite state transducer that $T_3 = T_2 \circ T_1$, which can be constructed as a machine
 4760 with one state for each pair of states in T_1 and T_2 (Mohri et al., 2002).

4761 **Example: Morphology and orthography** In English morphology, the suffix *-ed* is added
 4762 to signal the past tense for many verbs: *cook*→*cooked*, *want*→*wanted*, etc. However, English
 4763 **orthography** dictates that this process cannot produce a spelling with consecutive e’s, so
 4764 that *bake*→*baked*, not *bakeed*. A modular solution is to build separate transducers for mor-
 4765 phology and orthography. The morphological transducer T_M transduces from *bake+PAST*
 4766 to *bake+ed*, with the + symbol indicating a segment boundary. The input alphabet of T_M
 4767 includes the lexicon of words and the set of morphological features; the output alphabet
 4768 includes the characters *a-z* and the + boundary marker. Next, an orthographic transducer
 4769 T_O is responsible for the transductions *cook+ed*→*cooked*, and *bake+ed*→*baked*. The input
 4770 alphabet of T_O must be the same as the output alphabet for T_M , and the output alphabet

4771 is simply the characters *a-z*. The composed transducer ($T_O \circ T_M$) then transduces from
 4772 *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,

$$\delta(q_{m-1,i}, Y_m = j, q_{m,j}) = \log p(w_m, Y_m = j \mid Y_{m-1} = i) \quad [9.20]$$

$$= \log p(w_m \mid Y_m = j) + \log \Pr(Y_m = j \mid Y_{m-1} = i). \quad [9.21]$$

4773 Because there is only one possible transition for each tag Y_m , this WFSA is deterministic.
 4774 The score for any tag sequence $\{y_m\}_{m=1}^M$ is the sum of these log-probabilities, correspond-
 4775 ing to the total log probability $\log p(w, y)$. Furthermore, the trellis can be constructed by
 4776 the composition of simpler FSTs.

- 4777 • First, construct a “transition” transducer to represent a bigram probability model
 4778 over tag sequences, T_T . This transducer is almost identical to the n -gram language
 4779 model acceptor in § 9.1.3.1: there is one state for each tag, and the edge weights
 4780 equal to the transition log-probabilities, $\delta(q_i, j, j, q_j) = \log \Pr(Y_m = j \mid Y_{m-1} = i)$.
 4781 Note that T_T is a transducer, with identical input and output at each arc; this makes
 4782 it possible to compose T_T with other transducers.
- 4783 • Next, construct an “emission” transducer to represent the probability of words given
 4784 tags, T_E . This transducer has only a single state, with arcs for each word/tag pair,
 4785 $\delta(q_0, i, j, q_0) = \log \Pr(W_m = j \mid Y_m = i)$. The input vocabulary is the set of all tags,
 4786 and the output vocabulary is the set of all words.
- 4787 • The composition $T_E \circ T_T$ is a finite state transducer with one state per tag, as shown
 4788 in Figure 9.8. Each state has $V \times K$ outgoing edges, representing transitions to each
 4789 of the K other states, with outputs for each of the V words in the vocabulary. The
 4790 weights for these edges are equal to,

$$\delta(q_i, Y_m = j, w_m, q_j) = \log p(w_m, Y_m = j \mid Y_{m-1} = i). \quad [9.22]$$

- 4791 • The trellis is a structure with $M \times K$ nodes, for each of the M words to be tagged and
 4792 each of the K tags in the tagset. It can be built by composition of $(T_E \circ T_T)$ against an
 4793 unweighted **chain FSA** $M_A(w)$ that is specially constructed to accept only a given
 4794 input w_1, w_2, \dots, w_M , shown in Figure 9.9. The trellis for input w is built from the
 4795 composition $M_A(w) \circ (T_E \circ T_T)$. Composing with the unweighted $M_A(w)$ does not
 4796 affect the edge weights from $(T_E \circ T_T)$, but it selects the subset of paths that generate
 4797 the word sequence w .

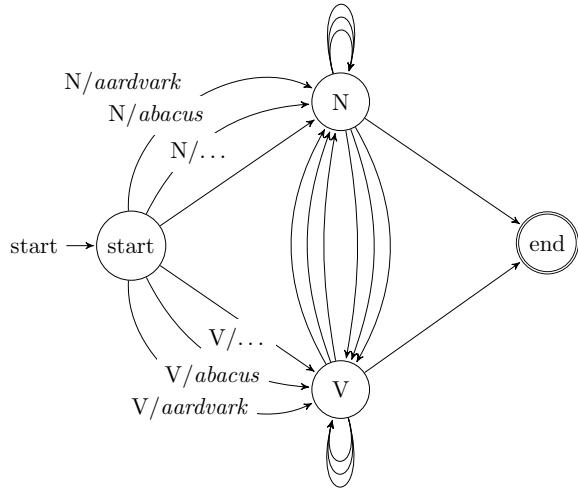


Figure 9.8: Finite state transducer for hidden Markov models, with a small tagset of nouns and verbs. For each pair of tags (including self-loops), there is an edge for every word in the vocabulary. For simplicity, input and output are only shown for the edges from the start state. Weights are also omitted 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 $\log \Pr(Y_m = \diamond \mid Y_{m-1} = i)$.

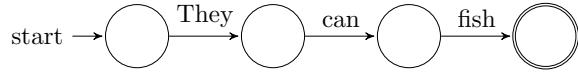


Figure 9.9: Chain finite state acceptor for the input *They can fish*.

4798 9.1.5 *Learning weighted finite state automata

4799 In generative models such as n -gram language models and hidden Markov models, the
 4800 edge weights correspond to log probabilities, which can be obtained from relative fre-
 4801 quency estimation. However, in other cases, we wish to learn the edge weights from in-
 4802 put/output pairs. This is difficult in non-deterministic finite state automata, because we
 4803 do not observe the specific arcs that are traversed in accepting the input, or in transducing
 4804 from input to output. The path through the automaton is a **latent variable**.

4805 Chapter 5 presented one method for learning with latent variables: expectation max-
 4806 imization (EM). This involves computing a distribution $q(\cdot)$ over the latent variable, and
 4807 iterating between updates to this distribution and updates to the parameters — in this
 4808 case, the arc weights. The **forward-backward algorithm** (§ 7.5.3.3) describes a dynamic
 4809 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 **latent 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

4845 glish includes center embedding, and so the argument goes, English grammar as a whole
 4846 cannot be regular.⁵

4847 A more practical argument for moving beyond regular languages is modularity. Many
 4848 linguistic phenomena — especially in syntax — involve constraints that apply at long
 4849 distance. Consider the problem of determiner-noun number agreement in English: we
 4850 can say *the coffee* and *these coffees*, but not **these coffee*. By itself, this is easy enough to model
 4851 in an FSA. However, fairly complex modifying expressions can be inserted between the
 4852 determiner and the noun:

- 4853 (9.5) the burnt coffee
- 4854 (9.6) the badly-ground coffee
- 4855 (9.7) the burnt and badly-ground Italian coffee
- 4856 (9.8) these burnt and badly-ground Italian coffees
- 4857 (9.9) *these burnt and badly-ground Italian coffee

4858 Again, an FSA can be designed to accept modifying expressions such as *burnt and badly-*
 4859 *ground Italian*. Let's call this FSA F_M . To reject the final example, a finite state acceptor
 4860 must somehow "remember" that the determiner was plural when it reaches the noun *cof-*
 4861 *fee* at the end of the expression. The only way to do this is to make two identical copies
 4862 of F_M : one for singular determiners, and one for plurals. While this is possible in the
 4863 finite state framework, it is inconvenient — especially in languages where more than one
 4864 attribute of the noun is marked by the determiner. **Context-free languages** facilitate mod-
 4865 ularity across such long-range dependencies.

4866 9.2.1 Context-free grammars

4867 Context-free languages are specified by **context-free grammars** (CFGs), which are tuples
 4868 (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

- 4869 • a finite set of **non-terminals** N ;
- 4870 • a finite alphabet Σ of **terminal symbols**;
- 4871 • a set of **production rules** R , each of the form $A \rightarrow \beta$, where $A \in N$ and $\beta \in (\Sigma \cup N)^*$;
- 4872 • a designated start symbol S .

4873 In the production rule $A \rightarrow \beta$, the left-hand side (LHS) A must be a non-terminal;
 4874 the right-hand side (RHS) can be a sequence of terminals or non-terminals, $\{n, \sigma\}^*, n \in$
 4875 $N, \sigma \in \Sigma$. A non-terminal can appear on the left-hand side of many production rules.
 4876 A non-terminal can appear on both the left-hand side and the right-hand side; this is a
 4877 **recursive production**, and is analogous to self-loops in finite state automata. The name
 4878 “context-free” is based on the property that the production rule depends only on the LHS,
 4879 and not on its ancestors or neighbors; this is analogous to Markov property of finite state
 4880 automata, in which the behavior at each step depends only on the current state, on not on
 4881 the path by which that state was reached.

4882 A **derivation** τ is a sequence of steps from the start symbol S to a surface string $w \in \Sigma^*$,
 4883 which is the **yield** of the derivation. A string w is in a context-free language if there is
 4884 some derivation from S yielding w . **Parsing** is the problem of finding a derivation for a
 4885 string in a grammar. Algorithms for parsing are described in chapter 10.

4886 Like regular expressions, context-free grammars define the language but not the com-
 4887 putation necessary to recognize it. The context-free analogues to finite state acceptors are
 4888 **pushdown automata**, a theoretical model of computation in which input symbols can be
 4889 pushed onto a stack with potentially infinite depth. For more details, see Sipser (2012).

4890 9.2.1.1 Example

4891 Figure 9.11 shows a context-free grammar for arithmetic expressions such as $1 + 2 \div 3 - 4$.
 4892 In this grammar, the terminal symbols include the digits $\{1, 2, \dots, 9\}$ and the op-
 4893 erators $\{+, -, \times, \div\}$. The rules include the $|$ symbol, a notational convenience that makes
 4894 it possible to specify multiple right-hand sides on a single line: the statement $A \rightarrow x | y$

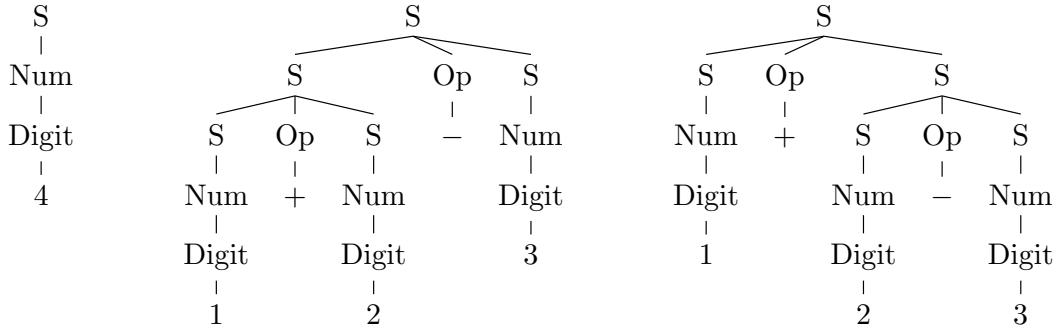


Figure 9.12: Some example derivations from the arithmetic grammar in Figure 9.11

4895 defines *two* productions, $A \rightarrow x$ and $A \rightarrow y$. This grammar is recursive: the non-termals S
4896 and NUM can produce themselves.

4897 Derivations are typically shown as trees, with production rules applied from the top
4898 to the bottom. The tree on the left in Figure 9.12 describes the derivation of a single digit,
4899 through the sequence of productions $S \rightarrow \text{NUM} \rightarrow \text{DIGIT} \rightarrow 4$ (these are all **unary produc-**
4900 **tions**, because the right-hand side contains a single element). The other two trees in
4901 Figure 9.12 show alternative derivations of the string $1 + 2 - 3$. The existence of multiple
4902 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:

$$(S (S (S (\text{Num} (Digit 1))) (\text{Op} +) (S (\text{Num} (Digit 2))))) (\text{Op} -) (S (\text{Num} (Digit 3)))) \quad [9.23]$$

$$(S (S (\text{Num} (Digit 1))) (\text{Op} +) (S (\text{Num} (Digit 2)) (\text{Op} -) (S (\text{Num} (Digit 3)))))). \quad [9.24]$$

4903 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 $n > 0$.

$$\begin{aligned} S &\rightarrow aSb \mid ab \\ S &\rightarrow aSb \mid aabb \mid ab \end{aligned}$$

4904 Two grammars are **weakly equivalent** if they generate the same strings. Two grammars
4905 are **strongly equivalent** if they generate the same strings via the same derivations. The
4906 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:

$$A \rightarrow BC$$

$$A \rightarrow a$$

4907 All CFGs can be converted into a CNF grammar that is weakly equivalent. To convert a
 4908 grammar into CNF, we first address productions that have more than two non-terminals
 4909 on the RHS by creating new “dummy” non-terminals. For example, if we have the pro-
 4910 duction,

$$W \rightarrow X Y Z, \quad [9.25]$$

it is replaced with two productions,

$$W \rightarrow X W \setminus X \quad [9.26]$$

$$W \setminus X \rightarrow Y Z. \quad [9.27]$$

4911 In these productions, $W \setminus X$ is a new dummy non-terminal. This transformation **binarizes**
 4912 the grammar, which is critical for efficient bottom-up parsing, as we will see in chapter 10.
 4913 Productions whose right-hand side contains a mix of terminal and non-terminal symbols
 4914 can be replaced in a similar fashion.

4915 Unary non-terminal productions $A \rightarrow B$ are replaced as follows: identify all produc-
 4916 tions $B \rightarrow \alpha$, and add $A \rightarrow \alpha$ to the grammar. For example, in the grammar described in
 4917 Figure 9.11, we would replace $\text{NUM} \rightarrow \text{DIGIT}$ with $\text{NUM} \rightarrow 1 \mid 2 \mid \dots \mid 9$. However, we
 4918 keep the production $\text{NUM} \rightarrow \text{NUM DIGIT}$, which is a valid binary production.

4919 9.2.2 Natural language syntax as a context-free language

4920 Context-free grammars can be used to represent **syntax**, which is the set of rules that
 4921 determine whether an utterance is judged to be grammatical. If this representation were
 4922 perfectly faithful, then a natural language such as English could be transformed into a
 4923 formal language, consisting of exactly the (infinite) set of strings that would be judged to
 4924 be grammatical by a fluent English speaker. We could then build parsing software that
 4925 would automatically determine if a given utterance were grammatical.⁷

4926 Contemporary theories generally do *not* consider natural languages to be context-free
 4927 (see § 9.3), yet context-free grammars are widely used in natural language parsing. The
 4928 reason is that context-free representations strike a good balance: they cover a broad range
 4929 of syntactic phenomena, and they can be parsed efficiently. This section therefore de-
 4930 scribes how to handle a core fragment of English syntax in context-free form, following

⁷To move beyond this cursory treatment of syntax, consult the short introductory manuscript by Bender (2013), or the longer text by Akmajian et al. (2010).

4931 the conventions of the **Penn Treebank** (PTB; Marcus et al., 1993), a large-scale annotation
 4932 of English language syntax. The generalization to “mildly” context-sensitive languages is
 4933 discussed in § 9.3.

4934 The Penn Treebank annotation is a **phrase-structure grammar** of English. This means
 4935 that sentences are broken down into **constituents**, which are contiguous sequences of
 4936 words that function as coherent units for the purpose of linguistic analysis. Constituents
 4937 generally have a few key properties:

4938 **Movement.** Constituents can often be moved around sentences as units.

- 4939 (9.10) Abigail gave (her brother) (a fish).
 4940 (9.11) Abigail gave (a fish) to (her brother).

4941 In contrast, *gave her* and *brother a* cannot easily be moved while preserving gram-
 4942 maticality.

4943 **Substitution.** Constituents can be substituted by other phrases of the same type.

- 4944 (9.12) Max thanked (his older sister).
 4945 (9.13) Max thanked (her).

4946 In contrast, substitution is not possible for other contiguous units like *Max thanked*
 4947 and *thanked his*.

4948 **Coordination.** Coordinators like *and* and *or* can conjoin constituents.

- 4949 (9.14) (Abigail) and (her younger brother) bought a fish.
 4950 (9.15) Abigail (bought a fish) and (gave it to Max).
 4951 (9.16) Abigail (bought) and (greedily ate) a fish.

4952 Units like *brother bought* and *bought a* cannot easily be coordinated.

4953 These examples argue for units such as *her brother* and *bought a fish* to be treated as con-
 4954 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
 4955 grammar, constituents are nested, so that *the senator from New Jersey* contains the con-
 4956 stituent *from New Jersey*, which in turn contains *New Jersey*. The sentence itself is the max-
 4957 imal constituent; each word is a minimal constituent, derived from a unary production
 4958 from a part-of-speech tag. Between part-of-speech tags and sentences are **phrases**. In
 4959 phrase-structure grammar, phrases have a type that is usually determined by their **head**
 4960 **word**: for example, a **noun phrase** corresponds to a noun and the group of words that

4962 modify it, such as *her younger brother*; a **verb phrase** includes the verb and its modifiers,
4963 such as *bought a fish* and *greedily ate it*.

4964 In context-free grammars, each phrase type is a non-terminal, and each constituent is
4965 the substring that the non-terminal yields. Grammar design involves choosing the right
4966 set of non-terminals. Fine-grained non-terminals make it possible to represent more fine-
4967 grained linguistic phenomena. For example, by distinguishing singular and plural noun
4968 phrases, it is possible to have a grammar of English that generates only sentences that
4969 obey subject-verb agreement. However, enforcing subject-verb agreement is considerably
4970 more complicated in languages like Spanish, where the verb must agree in both person
4971 and number with subject. In general, grammar designers must trade off between **over-**
4972 **generation** — a grammar that permits ungrammatical sentences — and **undergeneration**
4973 — a grammar that fails to generate grammatical sentences. Furthermore, if the grammar is
4974 to support manual annotation of syntactic structure, it must be simple enough to annotate
4975 efficiently.

4976 9.2.3 A phrase-structure grammar for English

4977 To better understand how phrase-structure grammar works, let's consider the specific
4978 case of the Penn Treebank grammar of English. The main phrase categories in the Penn
4979 Treebank (PTB) are based on the main part-of-speech classes: noun phrase (NP), verb
4980 phrase (VP), prepositional phrase (PP), adjectival phrase (ADJP), and adverbial phrase
4981 (ADVP). The top-level category is S, which conveniently stands in for both "sentence"
4982 and the "start" symbol. **Complement clauses** (e.g., *I take the good old fashioned ground that*
4983 *the whale is a fish*) are represented by the non-terminal SBAR. The terminal symbols in
4984 the grammar are individual words, which are generated from unary productions from
4985 part-of-speech tags (the PTB tagset is described in § 8.1).

4986 This section explores the productions from the major phrase-level categories, explaining
4987 how to generate individual tag sequences. The production rules are approached in a
4988 "theory-driven" manner: first the syntactic properties of each phrase type are described,
4989 and then some of the necessary production rules are listed. But it is important to keep
4990 in mind that the Penn Treebank was produced in a "data-driven" manner. After the set
4991 of non-terminals was specified, annotators were free to analyze each sentence in what-
4992 ever way seemed most linguistically accurate, subject to some high-level guidelines. The
4993 grammar of the Penn Treebank is simply the set of productions that were required to ana-
4994 lyze the several million words of the corpus. By design, the grammar overgenerates — it
4995 does not exclude ungrammatical sentences.

4996 **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:

$$S \rightarrow ADVP NP VP \quad \begin{matrix} Unfortunately Abigail ate the kimchi. \\ Abigail ate the kimchi and Max had a burger. \end{matrix} \quad [9.29]$$

$$S \rightarrow S CC S \quad \begin{matrix} Abigail ate the kimchi and Max had a burger. \\ Eat the kimchi. \end{matrix} \quad [9.30]$$

$$S \rightarrow VP \quad \begin{matrix} Eat the kimchi. \end{matrix} \quad [9.31]$$

4997 where ADVP is an adverbial phrase (e.g., *unfortunately*, *very unfortunately*) and CC is a
 4998 coordinating conjunction (e.g., *and*, *but*).⁸

4999 **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:

$$NP \rightarrow NN | NNS | NNP | PRP \quad [9.32]$$

$$NP \rightarrow DET NN | DET NNS | DET NNP \quad [9.33]$$

5000 The tags NN, NNS, and NNP refer to singular, plural, and proper nouns; PRP refers to
 5001 personal pronouns, and DET refers to determiners. The grammar also contains terminal
 5002 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.

5003 These multiple noun constructions can be combined with adjectival phrases and cardinal
 5004 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]

5005 These recursive productions are a major source of ambiguity, because the VP and PP non-
 5006 terminals can also generate NP children. Thus, the *the President of the Georgia Institute of*
 5007 *Technology* can be derived in two ways, as can *a whale taken near Shetland in October*.

5008 But aside from these few recursive productions, the noun phrase fragment of the Penn
 5009 Treebank grammar is relatively flat, containing a large of number of productions that go
 5010 from NP directly to a sequence of parts-of-speech. If noun phrases had more internal
 5011 structure, the grammar would need fewer rules, which, as we will see, would make pars-
 5012 ing faster and machine learning easier. Vadas and Curran (2011) propose to add additional
 5013 structure in the form of a new non-terminal called a **nominal modifier** (NML), e.g.,

5014 (9.17) (NP (NN crude) (NN oil) (NNS prices)) (PTB analysis)
 5015 (NP (NML (NN crude) (NN oil)) (NNS prices)) (NML-style analysis)

5016 Another proposal is to treat the determiner as the head of a **determiner phrase** (DP;
 5017 Abney, 1987). There are linguistic arguments for and against determiner phrases (e.g.,
 5018 Van Eynde, 2006). From the perspective of context-free grammar, DPs enable more struc-
 5019 tured analyses of some constituents, e.g.,

5020 (9.18) (NP (DT the) (JJ white) (NN whale)) (PTB analysis)
 5021 (DP (DT the) (NP (JJ white) (NN whale))) (DP-style analysis).

5022 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 (VBZ; *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

⁹This tagset is specific to English: for example, VBP is a meaningful category only because English morphology distinguishes third-person singular from all person-number combinations.

own:

$$VP \rightarrow VB \mid VBD \mid VBN \mid VBG \mid VBP \quad [9.41]$$

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]

- 5023 Each of these productions uses recursion, with the VP non-terminal appearing in both the
 5024 LHS and RHS. This enables the creation of complex verb phrases, such as *She will have*
 5025 *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]

- 5026 The first production overgenerates, licensing sentences like **Asha sees Boyang eats the kimchi*.
 5027 This problem could be addressed by designing a more specific set of sentence non-
 5028 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]

5029 Again, because these productions are not recursive, the grammar must include productions
 5030 for every verb part-of-speech.

A special set of verbs, known as **copula**, can take **predicative adjectives** as direct objects:

$VP \rightarrow VBZ\ ADJP$	<i>She is hungry</i>	[9.56]
$VP \rightarrow VBP\ ADJP$	<i>Success seems increasingly unlikely</i>	[9.57]

5031 The PTB does not have a special non-terminal for copular verbs, so this production generates
 5032 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]

5033 As the second production shows, particle productions are required for all configurations
 5034 of verb parts-of-speech and direct objects.

5035 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]

5036 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]

5037 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>	

5038 9.2.4 Grammatical ambiguity

5039 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 5040 such as information extraction (chapter 17) and sentence compression (Jing, 2000; Clarke 5041 and Lapata, 2008). However, the **ambiguity** of wide-coverage natural language grammars 5042 poses a serious problem for such potential applications. As an example, Figure 9.13 shows 5043 two possible analyses for the simple sentence *We eat sushi with chopsticks*, depending on 5044 whether the *chopsticks* modify *eat* or *sushi*. Realistic grammars can license thousands or 5045 even millions of parses for individual sentences. **Weighted context-free grammars** solve 5046 this problem by attaching weights to each production, and selecting the derivation with 5047 the highest score. This is the focus of chapter 10. 5048

5049 9.3 *Mildly context-sensitive languages

5050 Beyond context-free languages lie **context-sensitive languages**, in which the expansion 5051 of a non-terminal depends on its neighbors. In the general class of context-sensitive 5052 languages, computation becomes much more challenging: the membership problem for 5053 context-sensitive languages is PSPACE-complete. Since PSPACE contains the complexity 5054 class NP (problems that can be solved in polynomial time on a non-deterministic Turing

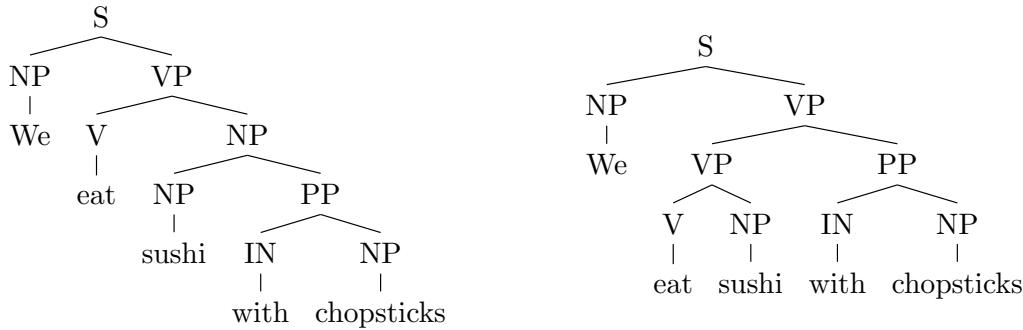


Figure 9.13: Two derivations of the same sentence

5055 machine), PSPACE-complete problems cannot be solved efficiently if $P \neq NP$. Thus, de-
 5056 signing an efficient parsing algorithm for the full class of context-sensitive languages is
 5057 probably hopeless.¹⁰

5058 However, Joshi (1985) identifies a set of properties that define **mildly context-sensitive**
 5059 **languages**, which are a strict subset of context-sensitive languages. Like context-free lan-
 5060 guages, mildly context-sensitive languages are efficiently parseable. However, the mildly
 5061 context-sensitive languages include non-context-free languages, such as the “copy lan-
 5062 guage” $\{ww \mid w \in \Sigma^*\}$ and the language $a^m b^n c^m d^n$. Both are characterized by **cross-**
 5063 **serial dependencies**, linking symbols at long distance across the string.¹¹ For example, in
 5064 the language $a^n b^m c^n d^m$, each a symbol is linked to exactly one c symbol, regardless of the
 5065 number of intervening b symbols.

5066 9.3.1 Context-sensitive phenomena in natural language

5067 Such phenomena are occasionally relevant to natural language. A classic example is found
 5068 in Swiss-German (Shieber, 1985), in which sentences such as *we let the children help Hans*
 5069 *paint the house* are realized by listing all nouns before all verbs, i.e., *we the children Hans the*
 5070 *house let help paint*. Furthermore, each noun’s determiner is dictated by the noun’s **case**
 5071 **marking** (the role it plays with respect to the verb). Using an argument that is analogous
 5072 to the earlier discussion of center-embedding (§ 9.2), Shieber describes these case marking
 5073 constraints as a set of cross-serial dependencies, homomorphic to $a^m b^n c^m d^n$, and therefore
 5074 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 $\{a^{2^n} \mid n \geq 0\}$.

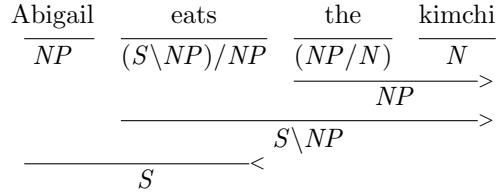


Figure 9.14: A syntactic analysis in CCG involving forward and backward function application

As with the move from regular to context-free languages, mildly context-sensitive languages can also 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). 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

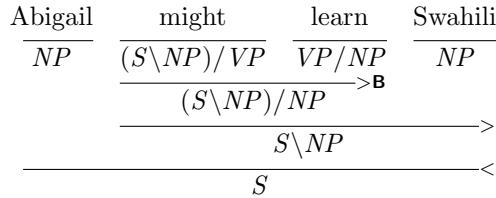


Figure 9.15: A syntactic analysis in CCG involving function composition (example modified from Steedman and Baldridge, 2011)

5104 its right results in a noun phrase, so determiners have the type NP/N. Figure 9.14 pro-
 5105 vides an example involving a transitive verb and a determiner. A key point from this
 5106 example is that it can be trivially transformed into phrase-structure tree, by treating each
 5107 function application as a constituent phrase. Indeed, when CCG's only combinatory op-
 5108 erators are forward and backward function application, it is equivalent to context-free
 5109 grammar. However, the location of the "effort" has changed. Rather than designing good
 5110 productions, the grammar designer must focus on the **lexicon** — choosing the right cate-
 5111 gories for each word. This makes it possible to parse a wide range of sentences using only
 5112 a few generic combinatory operators.

5113 Things become more interesting with the introduction of two additional operators:
 5114 **composition** and **type-raising**. Function composition enables the combination of com-
 5115 plex types: $X/Y \circ Y/Z \Rightarrow_B X/Z$ (forward composition) and $Y \setminus Z \circ X \setminus Y \Rightarrow_B X \setminus Z$ (back-
 5116 ward composition).¹² Composition makes it possible to "look inside" complex types, and
 5117 combine two adjacent units if the "input" for one is the "output" for the other. Figure 9.15
 5118 shows how function composition can be used to handle modal verbs. While this sen-
 5119 tence can be parsed using only function application, the composition-based analysis is
 5120 preferable because the unit *might learn* functions just like a transitive verb, as in the exam-
 5121 ple *Abigail studies Swahili*. This in turn makes it possible to analyze conjunctions such as
 5122 *Abigail studies and might learn Swahili*, attaching the direct object *Swahili* to the entire con-
 5123 joined verb phrase *studies and might learn*. The Penn Treebank grammar fragment from
 5124 § 9.2.3 would be unable to handle this case correctly: the direct object *Swahili* could attach
 5125 only to the second verb *learn*.

5126 Type raising converts an element of type X to a more complex type: $X \Rightarrow_T T/(T \setminus X)$
 5127 (forward type-raising to type T), and $X \Rightarrow_T T \setminus (T/X)$ (backward type-raising to type
 5128 T). Type-raising makes it possible to reverse the relationship between a function and its
 5129 argument — by transforming the argument into a function over functions over arguments!
 5130 An example may help. Figure 9.15 shows how to analyze an object relative clause, *a story*
 5131 *that Abigail tells*. The problem is that *tells* is a transitive verb, expecting a direct object to
 5132 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).

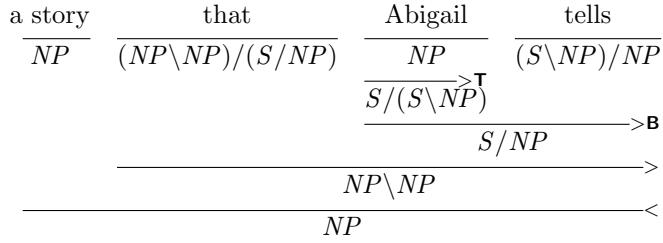


Figure 9.16: A syntactic analysis in CCG involving an object relative clause (based on slides from Alex Clark)

5133 *Abigail* from NP to the complex type $(S / NP) \setminus NP$. This function can then be combined
 5134 with the transitive verb *tells* by forward composition, resulting in the type (S / NP) , which
 5135 is a sentence lacking a direct object to its right.¹³ From here, we need only design the
 5136 lexical entry for the complementizer *that* to expect a right neighbor of type (S / NP) , and
 5137 the remainder of the derivation can proceed by function application.

5138 Composition and type-raising give CCG considerable power and flexibility, but at a price.
 5139 The simple sentence *Abigail tells Max* can be parsed in two different ways: by func-
 5140 tion application (first forming the verb phrase *tells Max*), and by type-raising and compo-
 5141 sition (first forming the non-constituent *Abigail tells*). This **derivational ambiguity** does
 5142 not affect the resulting linguistic analysis, so it is sometimes known as **spurious ambi-**
 5143 **guity**. Hockenmaier and Steedman (2007) present a translation algorithm for converting
 5144 the Penn Treebank into CCG derivations, using composition and type-raising only when
 5145 necessary.

5146 Exercises

- 5147 1. Sketch out the state diagram for finite-state acceptors for the following languages
 5148 on the alphabet $\{a, b\}$.
- 5149 a) Even-length strings. (Be sure to include 0 as an even number.)
- 5150 b) Strings that contain *aaa* as a substring.
- 5151 c) Strings containing an even number of *a* and an odd number of *b* symbols.
- 5152 d) Strings in which the substring *bbb* must be terminal if it appears — the string
 5153 need not contain *bbb*, but if it does, nothing can come after it.
- 5154 2. Levenshtein edit distance is the number of insertions, substitutions, or deletions
 5155 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.

- 5156 a) Define a finite-state acceptor that accepts all strings with edit distance 1 from
 5157 the target string, *target*.
 5158 b) Now think about how to generalize your design to accept all strings with edit
 5159 distance from the target string equal to d . If the target string has length ℓ , what
 5160 is the minimal number of states required?
- 5161 3. Construct an FSA in the style of Figure 9.3, which handles the following examples:
- 5162 • *nation*/N, *national*/ADJ, *nationalize*/V, *nationalizer*/N
 - 5163 • *America*/N, *American*/ADJ, *Americanize*/V, *Americanizer*/N
- 5164 Be sure that your FSA does not accept any further derivations, such as **nationalizeral*
 5165 and **Americanizern*.
- 5166 4. Show how to construct a trigram language model in a weighted finite-state acceptor.
 5167 Make sure that you handle the edge cases at the beginning and end of the input.
- 5168 5. Extend the FST in Figure 9.6 to handle the other two parts of rule 1a of the Porter
 5169 stemmer: *-sses* → *ss*, and *-ies* → *-i*.
- 5170 6. § 9.1.4.4 describes T_O , a transducer that captures English orthography by transduc-
 5171 ing *cook + ed* → *cooked* and *bake + ed* → *baked*. Design an unweighted finite-state
 5172 transducer that captures this property of English orthography.
 5173 Next, augment the transducer to appropriately model the suffix *-s* when applied to
 5174 words ending in *s*, e.g. *kiss+s* → *kisses*.
- 5175 7. Add parenthesization to the grammar in Figure 9.11 so that it is no longer ambigu-
 5176 ous.
- 5177 8. Construct three examples — a noun phrase, a verb phrase, and a sentence — which
 5178 can be derived from the Penn Treebank grammar fragment in § 9.2.3, yet are not
 5179 grammatical. Avoid reusing examples from the text. Optionally, propose corrections
 5180 to the grammar to avoid generating these cases.
- 5181 9. Produce parses for the following sentences, using the Penn Treebank grammar frag-
 5182 ment from § 9.2.3.
- 5183 (9.19) This aggression will not stand.
 5184 (9.20) I can get you a toe.
 5185 (9.21) Sometimes you eat the bar and sometimes the bar eats you.

5186 Then produce parses for three short sentences from a news article from this week.

5187 10. * One advantage of CCG is its flexibility in handling coordination:

5188 (9.22) *Abigail and Max speak Swahili*

5189 (9.23) *Abigail speaks and Max understands Swahili*

Define the lexical entry for *and* as

$$\text{and} := (X/X) \setminus X, \quad [9.77]$$

5190 where X can refer to any type. Using this lexical entry, show how to parse the two
5191 examples above. In the second example, *Swahili* should be combined with the coor-
5192 dination *Abigail speaks and Max understands*, and not just with the verb *understands*.

5193

Chapter 10

5194

Context-free parsing

5195 Parsing is the task of determining whether a string can be derived from a given context-
5196 free grammar, and if so, how. The parse structure can answer basic questions of who-did-
5197 what-to-whom, and is useful for various downstream tasks, such as semantic analysis
5198 (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.,

$$(((w_1 w_2) w_3) w_4) w_5), \quad (((w_1 (w_2 w_3)) w_4) w_5), \quad ((w_1 (w_2 (w_3 w_4))) w_5), \quad \dots$$

5199 The number of such bracketings is a **Catalan number**, which grows super-exponentially
5200 in the length of the sentence, $C_n = \frac{(2n)!}{(n+1)n!}$. As with sequence labeling, it is only possible to
5201 exhaustively search the space of parses by resorting to locality assumptions, which make it
5202 possible to search efficiently by reusing shared substructures with dynamic programming.
5203 This chapter focuses on a bottom-up dynamic programming algorithm, which enables
5204 exhaustive search of the space of possible parses, but imposes strict limitations on the
5205 form of scoring function. These limitations can be relaxed by abandoning exhaustive
5206 search. Non-exact search methods will be briefly discussed at the end of this chapter, and
5207 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

5208 **10.1 Deterministic bottom-up parsing**

5209 The **CKY algorithm**¹ is a bottom-up approach to parsing in a context-free grammar. It
 5210 efficiently tests whether a string is in a language, without enumerating all possible parses.
 5211 The algorithm first forms small constituents, and then tries to merge them into larger
 5212 constituents.

5213 To understand the algorithm, consider the input, *We eat sushi with chopsticks*. According-
 5214 ing to the toy grammar in Table 10.1, each terminal symbol can be generated by exactly
 5215 one unary production, resulting in the sequence NP V NP IN NP. The next step is to
 5216 try to apply binary productions to merge adjacent symbols into larger constituents: for
 5217 example, V NP can be merged into a verb phrase (VP), and IN NP can be merged into
 5218 a prepositional phrase (PP). Bottom-up parsing searches for a series of mergers that ulti-
 5219 mately results in the start symbol S covering the entire input.

5220 The CKY algorithm systematizes this search by incrementally constructing a table t in
 5221 which each cell $t[i, j]$ contains the set of nonterminals that can derive the span $w_{i+1:j}$. The
 5222 algorithm fills in the upper right triangle of the table; it begins with the diagonal, which
 5223 corresponds to substrings of length 1, and then computes derivations for progressively
 5224 larger substrings, until reaching the upper right corner $t[0, M]$, which corresponds to the
 5225 entire input, $w_{1:M}$. If the start symbol S is in $t[0, M]$, then the string w is in the language
 5226 defined by the grammar. This process is detailed in Algorithm 13, and the resulting data
 5227 structure is shown in Figure 10.1. Informally, here's how it works:

- 5228 • Begin by filling in the diagonal: the cells $t[m - 1, m]$ for all $m \in \{1, 2, \dots, M\}$. These
 5229 cells are filled with terminal productions that yield the individual tokens; for the
 5230 word $w_2 = \text{sushi}$, we fill in $t[1, 2] = \{\text{NP}\}$, and so on.
- 5231 • Then fill in the next diagonal, in which each cell corresponds to a subsequence of
 5232 length two: $t[0, 2], t[1, 3], \dots, t[M - 2, M]$. These cells are filled in by looking for
 5233 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  $X \rightarrow (\text{TRACEBACK}(Y, i, k, b), \text{TRACEBACK}(Z, k, j, b))$ 

```

5234 sponding to left and right children. For example, the cell $t[1, 3]$ includes VP because
 5235 the grammar includes the production $\text{VP} \rightarrow \text{V NP}$, and the chart contains $\text{V} \in t[1, 2]$
 5236 and $\text{NP} \in t[2, 3]$.

- 5237 • At the next diagonal, the entries correspond to spans of length three. At this level,
 5238 there is an additional decision at each cell: where to split the left and right children.
 5239 The cell $t[i, j]$ corresponds to the subsequence $w_{i+1:j}$, and we must choose some
 5240 *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
 5241 consider all possible k , looking for productions that generate elements in $t[i, k]$ and
 5242 $t[k, j]$; the left-hand side of all such productions can be added to $t[i, j]$. When it is
 5243 time to compute $t[i, j]$, the cells $t[i, k]$ and $t[k, j]$ are guaranteed to be complete, since
 5244 these cells correspond to shorter sub-strings of the input.
- 5245 • The process continues until we reach $t[0, M]$.

5246 Figure 10.1 shows the chart that arises from parsing the sentence *We eat sushi with chop-*
 5247 *sticks* using the grammar defined above.

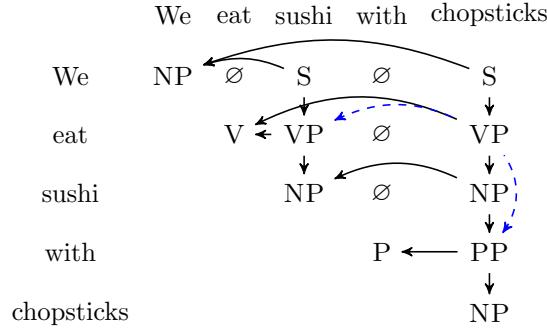


Figure 10.1: An example completed CKY chart. The solid and dashed lines show the back pointers resulting from the two different derivations of VP in position $t[1, 5]$.

5248 10.1.1 Recovering the parse tree

5249 As with the Viterbi algorithm, it is possible to identify a successful parse by storing and
 5250 traversing an additional table of back-pointers. If we add an entry X to cell $t[i, j]$ by using
 5251 the production $X \rightarrow YZ$ and the split point k , then we store the back-pointer $b[i, j, X] =$
 5252 (Y, Z, k) . Once the table is complete, we can recover a parse by tracing this pointers,
 5253 starting at $b[0, M, S]$, and stopping when they ground out at terminal productions.

5254 For ambiguous sentences, there will be multiple paths to reach $S \in t[0, M]$. For exam-
 5255 ple, in Figure 10.1, the goal state $S \in t[0, M]$ is reached through the state $VP \in t[1, 5]$, and
 5256 there are two different ways to generate this constituent: one with *(eat sushi)* and *(with
 5257 chopsticks)* as children, and another with *(eat)* and *(sushi with chopsticks)* as children. The
 5258 presence of multiple paths indicates that the input can be generated by the grammar in
 5259 more than one way. In Algorithm 13, one of these derivations is selected arbitrarily. As
 5260 discussed in § 10.3, **weighted context-free grammars** can select a single parse that maxi-
 5261 mizes a scoring function.

5262 10.1.2 Non-binary productions

5263 The CKY algorithm assumes that all productions with non-terminals on the right-hand
 5264 side (RHS) are binary. But in real grammars, such as the one considered in chapter 9,
 5265 there will be productions with more than two elements on the right-hand side, and other
 5266 productions with only a single element.

- 5267 • Productions with more than two elements on the right-hand side can be **binarized**
 5268 by creating additional non-terminals, as described in § 9.2.1.2. For example, given
 5269 the production $VP \rightarrow V NP NP$ (for ditransitive verbs), we can convert to $VP \rightarrow$
 5270 $VP_{ditrans}/NP NP$, 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.

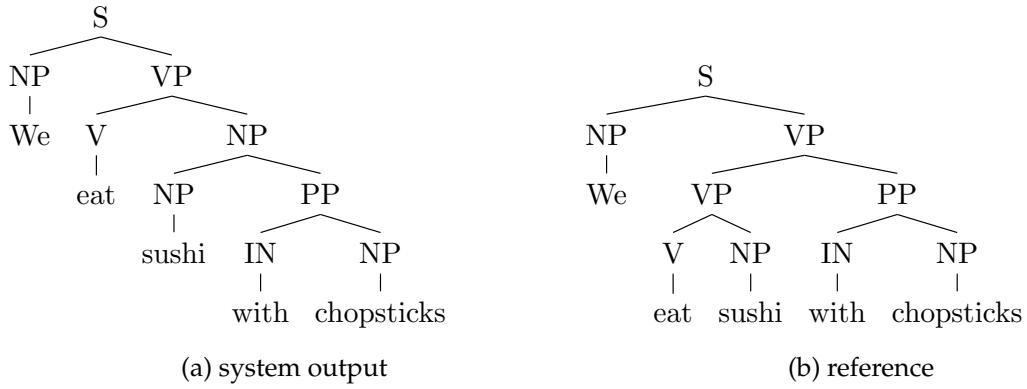


Figure 10.2: Two possible analyses from the grammar in Table 10.1

5333 The labeled and unlabeled precision of this parse is $\frac{3}{4} = 0.75$, and the recall is $\frac{3}{4} = 0.75$, for
 5334 an F-measure of 0.75. For an example in which precision and recall are not equal, suppose
 5335 the reference parse instead included the production $VP \rightarrow V NP PP$. In this parse, the
 5336 reference does not contain the constituent $w_{2:3}$, so the recall would be 1.³

5337 10.2.2 Local solutions

5338 Some ambiguity can be resolved locally. Consider the following examples,

5339 (10.1) We met the President on Monday.

5340 (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,

$$p(on \mid met) \geq p(on \mid President) \quad [10.1]$$

$$p(of \mid met) \geq p(of \mid President). \quad [10.2]$$

5341 A comparison of these probabilities would successfully resolve this case (Hindle and
5342 Rooth, 1993). Other cases, such as the example ... *eat sushi with chopsticks*, require consider-
5343 ing the object of the preposition — consider the alternative ... *eat sushi with soy sauce*. With
5344 sufficient labeled data, the problem of prepositional phrase attachment can be treated as
5345 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.

5346 However, there are inherent limitations to local solutions. While toy examples may
 5347 have just a few ambiguities to resolve, realistic sentences have thousands or millions of
 5348 possible parses. Furthermore, attachment decisions are interdependent, as shown in the
 5349 garden path example:

5350 (10.3) Cats scratch people with claws with knives.

5351 We may want to attach *with claws* to *scratch*, as would be correct in the shorter sentence
 5352 in *cats scratch people with claws*. But this leaves nowhere to attach *with knives*. The cor-
 5353 rect interpretation can be identified only by considering the attachment decisions jointly.
 5354 The huge number of potential parses may seem to make exhaustive search impossible.
 5355 But as with sequence labeling, locality assumptions make it possible to search this space
 5356 efficiently.

5357 10.3 Weighted Context-Free Grammars

5358 Let us define a derivation τ as a set of **anchored productions**,

$$\tau = \{X \rightarrow \alpha, (i, j, k)\}, \quad [10.3]$$

5359 with X corresponding to the left-hand side non-terminal and α corresponding to the right-
 5360 hand side. For grammars in Chomsky normal form, α is either a pair of non-terminals or
 5361 a terminal symbol. The indices i, j, k anchor the production in the input, with X deriving
 5362 the span $w_{i+1:j}$. For binary productions, $w_{i+1:k}$ indicates the span of the left child, and
 5363 $w_{k+1:j}$ indicates the span of the right child; for unary productions, k is ignored. For an
 5364 input w , the optimal parse is then,

$$\hat{\tau} = \underset{\tau \in \mathcal{T}(w)}{\operatorname{argmax}} \Psi(\tau), \quad [10.4]$$

5365 where $\mathcal{T}(w)$ is the set of derivations that yield the input w .

5366 The scoring function Ψ decomposes across anchored productions,

$$\Psi(\tau) = \sum_{(X \rightarrow \alpha, (i, j, k)) \in \tau} \psi(X \rightarrow \alpha, (i, j, k)). \quad [10.5]$$

5367 This is a locality assumption, akin to the assumption in Viterbi sequence labeling. In this
 5368 case, the assumption states that the overall score is a sum over scores of productions,
 5369 which are computed independently. In a **weighted context-free grammar** (WCFG), the
 5370 score of each anchored production $X \rightarrow (\alpha, i, j, k)$ is simply $\psi(X \rightarrow \alpha)$, ignoring the
 5371 anchors (i, j, k) . In other parsing models, the anchors can be used to access features of the
 5372 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.

$$\begin{aligned} \Psi(\tau) &= \psi(S \rightarrow \text{NP VP}) + \psi(VP \rightarrow \text{VP PP}) + \psi(VP \rightarrow \text{V NP}) + \psi(PP \rightarrow \text{IN NP}) \\ &\quad + \psi(NP \rightarrow \text{We}) + \psi(V \rightarrow \text{eat}) + \psi(NP \rightarrow \text{sushi}) + \psi(IN \rightarrow \text{with}) + \psi(NP \rightarrow \text{chopsticks}) \end{aligned} \quad [10.6]$$

$$= 0 - 2 - 1 + 0 - 2 + 0 - 3 + 0 - 3 = -11. \quad [10.7]$$

5373 In the alternative parse in Figure 10.2a, the production $VP \rightarrow VP PP$ (with score -2) is
 5374 replaced with the production $NP \rightarrow NP PP$ (with score -1); all other productions are the
 5375 same. As a result, the score for this parse is -10.

5376 This example hints at a big problem with WCFG parsing on non-terminals such as
 5377 NP, VP, and PP: a WCFG will *always* prefer either VP or NP attachment, without regard
 5378 to what is being attached! This problem is addressed in § 10.5.

5379 10.3.1 Parsing with weighted context-free grammars

5380 The optimization problem in Equation 10.4 can be solved by modifying the CKY algo-
 5381 rithm. In the deterministic CKY algorithm, each cell $t[i, j]$ stored a set of non-terminals
 5382 capable of deriving the span $w_{i+1:j}$. We now augment the table so that the cell $t[i, j, X]$
 5383 is the *score of the best derivation of $w_{i+1:j}$ from non-terminal X* . This score is computed
 5384 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( $S, 0, M, b$ )

```

- 5385 • the score of the anchored production, $\psi(X \rightarrow Y Z, (i, j, k))$;
- 5386 • the score of the best derivation of the left child, $t[i, k, Y]$;
- 5387 • the score of the best derivation of the right child, $t[k, j, Z]$.

5388 These scores are combined by addition. As in the unscored CKY algorithm, the table
 5389 is constructed by considering spans of increasing length, so the scores for spans $t[i, k, Y]$
 5390 and $t[k, j, Z]$ are guaranteed to be available at the time we compute the score $t[i, j, X]$. The
 5391 value $t[0, M, S]$ is the score of the best derivation of w from the grammar. Algorithm 14
 5392 formalizes this procedure.

5393 As in unweighted CKY, the parse is recovered from the table of back pointers b , where
 5394 each $b[i, j, X]$ stores the argmax split point k and production $X \rightarrow Y Z$ in the derivation of
 5395 $w_{i+1:j}$ from X . The best parse can be obtained by tracing these pointers backwards from
 5396 $b[0, M, S]$, all the way to the terminal symbols. This is analogous to the computation of the
 5397 best sequence of labels in the Viterbi algorithm by tracing pointers backwards from the
 5398 end of the trellis. Note that we need only store back-pointers for the *best* path to $t[i, j, X]$;
 5399 this follows from the locality assumption that the global score for a parse is a combination
 5400 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,

$$t[1, 5, \text{VP}] = \max(\psi(\text{VP} \rightarrow \text{VP PP}, (1, 3, 5)) + t[1, 3, \text{VP}] + t[3, 5, \text{PP}], \\ \psi(\text{VP} \rightarrow \text{V NP}, (1, 2, 5)) + t[1, 2, \text{V}] + t[2, 5, \text{NP}]) \quad [10.8]$$

$$= \max(-2 - 4 - 3, -1 + 0 - 7) = -8. \quad [10.9]$$

5401 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.

5403 **10.3.2 Probabilistic context-free grammars**

5404 **Probabilistic context-free grammars (PCFGs)** are a special case of weighted context-
5405 free grammars that arises when the weights correspond to probabilities. Specifically, the
5406 weight $\psi(X \rightarrow \alpha, (i, j, k)) = \log p(\alpha | X)$, where the probability of the right-hand side
5407 α is conditioned on the non-terminal X . These probabilities must be normalized over all
5408 possible right-hand sides, so that $\sum_\alpha p(\alpha | X) = 1$, for all X . For a given parse τ , the prod-
5409 uct of the probabilities of the productions is equal to $p(\tau)$, under the **generative model**
5410 $\tau \sim \text{DRAWSUBTREE}(S)$, where the function DRAWSUBTREE is defined in Algorithm 15.

5411 The conditional probability of a parse given a string is,

$$p(\tau | w) = \frac{p(\tau)}{\sum_{\tau' \in \mathcal{T}(w)} p(\tau')} = \frac{\exp \Psi(\tau)}{\sum_{\tau' \in \mathcal{T}(w)} \exp \Psi(\tau')}, \quad [10.10]$$

5412 where $\Psi(\tau) = \sum_{X \rightarrow \alpha, (i, j, k) \in \tau} \psi(X \rightarrow \alpha)$; the anchor is ignored. Because the probability
5413 is monotonic in the score $\Psi(\tau)$, the maximum likelihood parse can be identified by the
5414 CKY algorithm without modification. If a normalized probability $p(\tau | w)$ is required,
5415 the denominator of Equation 10.10 can be computed by the **inside recurrence**, described
5416 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,

$$p(\tau_1 | \mathbf{w}) = \frac{p(\tau_1)}{p(\tau_1) + p(\tau_2)} = \frac{\exp \Psi(\tau_1)}{\exp \Psi(\tau_1) + \exp \Psi(\tau_2)} = \frac{2^{-10}}{2^{-10} + 2^{-11}} = \frac{2}{3}. \quad [10.11]$$

5417 **The inside recurrence** The denominator of Equation 10.10 can be viewed as a language
5418 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:

$$t[i, j, X] = \log \sum_{k, Y, Z} \exp (\psi(X \rightarrow Y Z) + t[i, k, Y] + t[k, j, Z]) \quad [10.13]$$

$$= \log \sum_{k, Y, Z} \exp (\log p(Y Z | X) + \log p(Y \rightarrow \mathbf{w}_{i+1:k}) + \log p(Z \rightarrow \mathbf{w}_{k+1:j})) \quad [10.14]$$

$$= \log \sum_{k, Y, Z} p(Y Z | X) \times p(Y \rightarrow \mathbf{w}_{i+1:k}) \times p(Z \rightarrow \mathbf{w}_{k+1:j}) \quad [10.15]$$

$$= \log \sum_{k, Y, Z} p(Y Z, \mathbf{w}_{i+1:k}, \mathbf{w}_{k+1:j} | X) \quad [10.16]$$

$$= \log p(X \rightarrow \mathbf{w}_{i+1:j}). \quad [10.17]$$

5419 This is called the **inside recurrence**, because it computes the probability of each subtree
5420 as a combination of the probabilities of the smaller subtrees that are inside of it. The
5421 name implies a corresponding **outside recurrence**, which computes the probability of
5422 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
5423 recurrence is described in § 10.4.3. The inside and outside recurrences are analogous to the
5424 forward and backward recurrences in probabilistic sequence labeling (see § 7.5.3.3). They
5425 can be used to compute the marginal probabilities of individual anchored productions,
5426 $p(X \rightarrow \alpha, (i, j, k) | \mathbf{w})$, summing over all possible derivations of \mathbf{w} .

5427 **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:

$$t[i, j, X] = \bigoplus_{k, Y, Z} \psi(X \rightarrow Y Z, (i, j, k)) \otimes t[i, k, Y] \otimes t[k, j, Z]. \quad [10.18]$$

5428 This recurrence subsumes all of the algorithms that we have encountered in this chapter.

5429 **Unweighted CKY.** When $\psi(X \rightarrow \alpha, (i, j, k))$ is a *Boolean truth value* $\{\top, \perp\}$, \otimes is logical
5430 conjunction, and \bigoplus is logical disjunction, then we derive the CKY recurrence for
5431 unweighted context-free grammars, discussed in § 10.1 and Algorithm 13.

5432 **Weighted CKY.** When $\psi(X \rightarrow \alpha, (i, j, k))$ is a scalar score, \otimes is addition, and \bigoplus is maxi-
5433 mization, then we derive the CKY recurrence for weighted context-free grammars,
5434 discussed in § 10.3 and Algorithm 14. When $\psi(X \rightarrow \alpha, (i, j, k)) = \log p(\alpha \mid X)$,
5435 this same setting derives the CKY recurrence for finding the maximum likelihood
5436 derivation in a probabilistic context-free grammar.

5437 **Inside recurrence.** When $\psi(X \rightarrow \alpha, (i, j, k))$ is a log probability, \otimes is addition, and $\bigoplus =$
5438 $\log \sum \exp$, then we derive the inside recurrence for probabilistic context-free gram-
5439 mmars, discussed in § 10.3.2. It is also possible to set $\psi(X \rightarrow \alpha, (i, j, k))$ directly equal
5440 to the probability $p(\alpha \mid X)$. In this case, \otimes is multiplication, and \bigoplus is addition.
5441 While this may seem more intuitive than working with log probabilities, there is the
5442 risk of underflow on long inputs.

5443 Regardless of how the scores are combined, the key point is the locality assumption:
5444 the score for a derivation is the combination of the independent scores for each anchored
5445 production, and these scores do not depend on any other part of the derivation. For exam-
5446 ple, if two non-terminals are siblings, the scores of productions from these non-terminals
5447 are computed independently. This locality assumption is analogous to the first-order
5448 Markov assumption in sequence labeling, where the score for transitions between tags
5449 depends only on the previous tag and current tag, and not on the history. As with se-
5450 quence labeling, this assumption makes it possible to find the optimal parse efficiently; its
5451 linguistic limitations are discussed in § 10.5.

5452 **10.4 Learning weighted context-free grammars**

5453 Like sequence labeling, context-free parsing is a form of structure prediction. As a result,
5454 WCFGs can be learned using the same set of algorithms: generative probabilistic models,
5455 structured perceptron, maximum conditional likelihood, and maximum margin learning.

5456 In all cases, learning requires a **treebank**, which is a dataset of sentences labeled with
 5457 context-free parses. Parsing research was catalyzed by the **Penn Treebank** (Marcus et al.,
 5458 1993), the first large-scale dataset of this type (see § 9.2.2). Phrase structure treebanks exist
 5459 for roughly two dozen other languages, with coverage mainly restricted to European and
 5460 East Asian languages, plus Arabic and Urdu.

5461 **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:

$$\psi(X \rightarrow \alpha) = \log p(X \rightarrow \alpha) \quad [10.19]$$

$$\hat{p}(X \rightarrow \alpha) = \frac{\text{count}(X \rightarrow \alpha)}{\text{count}(X)}. \quad [10.20]$$

5462 For example, the probability of the production $NP \rightarrow DET\ NN$ is the corpus count of
 5463 this production, divided by the count of the non-terminal NP . This estimator applies
 5464 to terminal productions as well: the probability of $NN \rightarrow whale$ is the count of how often
 5465 *whale* appears in the corpus as generated from an NN tag, divided by the total count of the
 5466 NN tag. Even with the largest treebanks — currently on the order of one million tokens
 5467 — it is difficult to accurately compute probabilities of even moderately rare events, such
 5468 as $NN \rightarrow whale$. Therefore, smoothing is critical for making PCFGs effective.

5469 **10.4.2 Feature-based parsing**

5470 The scores for each production can be computed as an inner product of weights and fea-
 5471 tures,

$$\psi(X \rightarrow \alpha) = \boldsymbol{\theta} \cdot \mathbf{f}(X, \alpha, (i, j, k), \mathbf{w}), \quad [10.21]$$

5472 where the feature vector $\mathbf{f}(X, \alpha)$ is a function of the left-hand side X , the right-hand side
 5473 α , the anchor indices (i, j, k) , and the input \mathbf{w} .

5474 The basic feature $\mathbf{f}(X, \alpha, (i, j, k)) = \{(X, \alpha)\}$ encodes only the identity of the pro-
 5475 duction itself, which is a discriminatively-trained model with the same expressiveness as
 5476 a PCFG. Features on anchored productions can include the words that border the span
 5477 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
 5478 span $w_{i+1:k}$, and so on (Durrett and Klein, 2015). Scores on anchored productions can be
 5479 incorporated into CKY parsing without any modification to the algorithm, because it is
 5480 still possible to compute each element of the table $t[i, j, X]$ recursively from its immediate
 5481 children.

5482 Other features can be obtained by grouping elements on either the left-hand or right-
 5483 hand side: for example it can be particularly beneficial to compute additional features
 5484 by clustering terminal symbols, with features corresponding to groups of words with
 5485 similar syntactic properties. The clustering can be obtained from unlabeled datasets that
 5486 are much larger than any treebank, improving coverage. Such methods are described in
 5487 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),

$$\mathbf{f}(\tau, \mathbf{w}^{(i)}) = \sum_{(X \rightarrow \alpha, (i, j, k)) \in \tau} \mathbf{f}(X, \alpha, (i, j, k), \mathbf{w}^{(i)}) \quad [10.22]$$

$$\hat{\tau} = \operatorname{argmax}_{\tau \in \mathcal{T}(\mathbf{w})} \mathbf{f}(\tau, \mathbf{w}^{(i)}) \quad [10.23]$$

$$\boldsymbol{\theta} \leftarrow \mathbf{f}(\tau^{(i)}, \mathbf{w}^{(i)}) - \mathbf{f}(\hat{\tau}, \mathbf{w}^{(i)}). \quad [10.24]$$

5488 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,
 5489 where Δ is a non-negative cost function, such as the Hamming loss (Stern et al., 2017). It
 5490 is also possible to train feature-based parsing models by conditional log-likelihood, as
 5491 described in the next section.

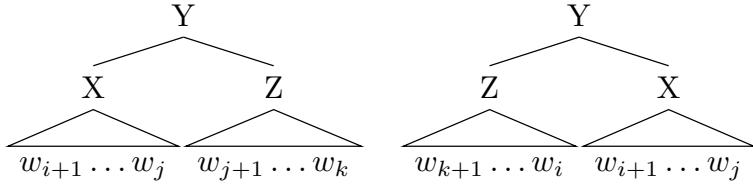
5493 10.4.3 *Conditional random field parsing

5494 The score of a derivation $\Psi(\tau)$ can be converted into a probability by normalizing over all
 5495 possible derivations,

$$p(\tau | \mathbf{w}) = \frac{\exp \Psi(\tau)}{\sum_{\tau' \in \mathcal{T}(\mathbf{w})} \exp \Psi(\tau')}. \quad [10.25]$$

5496 Using this probability, a WCFG can be trained by maximizing the conditional log-likelihood
 5497 of a labeled corpus.

5498 Just as in logistic regression and the conditional random field over sequences, the
 5499 gradient of the conditional log-likelihood is the difference between the observed and ex-
 5500 pected counts of each feature. The expectation $E_{\tau|\mathbf{w}}[\mathbf{f}(\tau, \mathbf{w}^{(i)}); \boldsymbol{\theta}]$ requires summing over
 5501 all possible parses, and computing the marginal probabilities of anchored productions,
 5502 $p(X \rightarrow \alpha, (i, j, k) | \mathbf{w})$. In CRF sequence labeling, marginal probabilities over tag bigrams
 5503 are computed by the two-pass **forward-backward algorithm** (§ 7.5.3.3). The analogue for
 5504 context-free grammars is the **inside-outside algorithm**, in which marginal probabilities
 5505 are computed from terms generated by an upward and downward pass over the parsing
 5506 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,

$$\alpha(i, j, X) \triangleq \log \sum_{(X \rightarrow Y \ Z)} \sum_{k=i+1}^j \exp (\psi(X \rightarrow Y \ Z, (i, j, k)) + \alpha(i, k, Y) + \alpha(k, j, Z)). \quad [10.26]$$

5507 The initial condition of this recurrence is $\alpha(m - 1, m, X) = \psi(X \rightarrow w_m)$. The de-
5508 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,

$$\exp \beta(i, j, X) \triangleq \sum_{(Y \rightarrow X \ Z)} \sum_{k=j+1}^M \exp [\psi(Y \rightarrow X \ Z, (i, k, j)) + \alpha(j, k, Z) + \beta(i, k, Y)] \quad [10.27]$$

$$+ \sum_{(Y \rightarrow Z \ X)} \sum_{k=0}^{i-1} \exp [\psi(Y \rightarrow Z \ X, (k, i, j)) + \alpha(k, i, Z) + \beta(k, j, Y)]. \quad [10.28]$$

5509 The first line of Equation 10.28 is the score under the condition that X is a left child
5510 of its parent, which spans $w_{i+1:k}$, with $k > j$; the second line is the score under the
5511 condition that X is a right child of its parent Y , which spans $w_{k+1:j}$, with $k < i$.
5512 The two cases are shown in Figure 10.3. In each case, we sum over all possible
5513 productions with X on the right-hand side. The parent Y is bounded on one side

5514 by either i or j , depending on whether X is a left or right child of Y ; we must sum
 5515 over all possible values for the other boundary. The initial conditions for the outside
 5516 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,

$$p(X \rightsquigarrow w_{i+1:j} | w) = \frac{p(X \rightsquigarrow w_{i+1:j}, w)}{p(w)} \quad [10.29]$$

$$= \frac{p(w_{i+1:j} | X) \times p(X, w_{1:i}, x_{j+1:M})}{p(w)} \quad [10.30]$$

$$= \frac{\exp(\alpha(i, j, X) + \beta(i, j, X))}{\exp \alpha(0, M, S)}. \quad [10.31]$$

5517 Marginal probabilities of individual productions can be computed similarly (see exercise
 5518 2). These marginal probabilities can be used for training a conditional random field parser,
 5519 and also for the task of unsupervised **grammar induction**, in which a PCFG is estimated
 5520 from a dataset of unlabeled text (Lari and Young, 1990; Pereira and Schabes, 1992).

5521 10.4.4 Neural context-free grammars

5522 Recent work has applied neural representations to parsing, representing each span with
 5523 a dense numerical vector (Socher et al., 2013; Durrett and Klein, 2015; Cross and Huang,
 5524 2016).⁴ For example, the anchor (i, j, k) and sentence w can be associated with a fixed-
 5525 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),

$$\tilde{\mathbf{v}}_{(i,j,k)} = \text{FeedForward}(\mathbf{v}_{(i,j,k)}) \quad [10.33]$$

$$\psi(X \rightarrow \alpha, (i, j, k)) = \tilde{\mathbf{v}}_{(i,j,k)}^\top \Theta \mathbf{f}(X \rightarrow \alpha), \quad [10.34]$$

5526 where $\mathbf{f}(X \rightarrow \alpha)$ is a vector of discrete features of the production, and Θ is a parameter
 5527 matrix. The matrix Θ and the parameters of the feedforward network can be learned by
 5528 backpropagating from an objective such as the margin loss or the negative conditional
 5529 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).

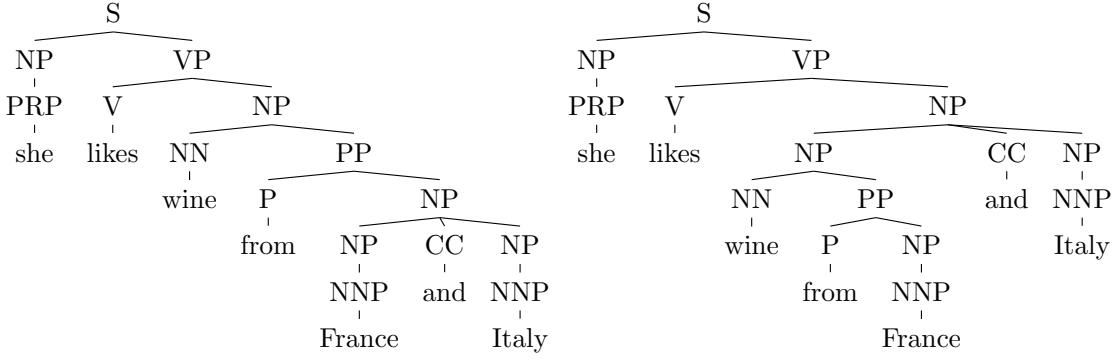


Figure 10.4: The left parse is preferable because of the conjunction of phrases headed by *France* and *Italy*, but these parses cannot be distinguished by a WCFG.

5530 10.5 Grammar refinement

5531 The locality assumptions underlying CFG parsing depend on the granularity of the non-
 5532 terminals. For the Penn Treebank non-terminals, there are several reasons to believe that
 5533 these assumptions are too strong to enable accurate parsing (Johnson, 1998):

- 5534 • The context-free assumption is too strict: for example, the probability of the produc-
 5535 tion $NP \rightarrow NP\ PP$ is much higher (in the PTB) if the parent of the noun phrase is a
 5536 verb phrase (indicating that the NP is a direct object) than if the parent is a sentence
 5537 (indicating that the NP is the subject of the sentence).
- 5538 • The Penn Treebank non-terminals are too coarse: there are many kinds of noun
 5539 phrases and verb phrases, and accurate parsing sometimes requires knowing the
 5540 difference. As we have already seen, when faced with prepositional phrase at-
 5541 tachment ambiguity, a weighted CFG will either always choose NP attachment (if
 5542 $\psi(NP \rightarrow NP\ PP) > \psi(VP \rightarrow VP\ PP)$), or it will always choose VP attachment. To
 5543 get more nuanced behavior, more fine-grained non-terminals are needed.
- 5544 • More generally, accurate parsing requires some amount of **semantics** — understand-
 5545 ing the meaning of the text to be parsed. Consider the example *cats scratch people with*
 5546 *claws*: knowledge of about *cats*, *claws*, and scratching is necessary to correctly resolve
 5547 the attachment ambiguity.

5548 An extreme example is shown in Figure 10.4. The analysis on the left is preferred
 5549 because of the conjunction of similar entities *France* and *Italy*. But given the non-terminals
 5550 shown in the analyses, there is no way to differentiate these two parses, since they include
 5551 exactly the same productions. What is needed seems to be more precise non-terminals.
 5552 One possibility would be to rethink the linguistics behind the Penn Treebank, and ask

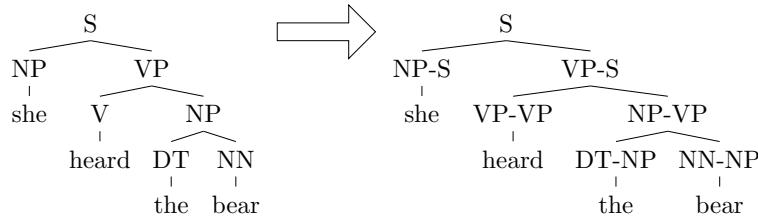


Figure 10.5: Parent annotation in a CFG derivation

5553 the annotators to try again. But the original annotation effort took five years, and there
 5554 is a little appetite for another annotation effort of this scope. Researchers have therefore
 5555 turned to automated techniques.

5556 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):

$$\Pr(\text{NP} \rightarrow \text{NP PP}) = 11\% \quad [10.35]$$

$$\Pr(\text{NP under S} \rightarrow \text{NP PP}) = 9\% \quad [10.36]$$

$$\Pr(\text{NP under VP} \rightarrow \text{NP PP}) = 23\%. \quad [10.37]$$

5557 This phenomenon can be captured by **parent annotation** (Johnson, 1998), in which each
 5558 non-terminal is augmented with the identity of its parent, as shown in Figure 10.5). This is
 5559 sometimes called **vertical Markovization**, since a Markov dependency is introduced be-
 5560 tween each node and its parent (Klein and Manning, 2003). It is analogous to moving from
 5561 a bigram to a trigram context in a hidden Markov model. In principle, parent annotation
 5562 squares the size of the set of non-terminals, which could make parsing considerably less
 5563 efficient. But in practice, the increase in the number of non-terminals that actually appear
 5564 in the data is relatively modest (Johnson, 1998).

5565 Parent annotation weakens the WCFG locality assumptions. This improves accuracy
 5566 by enabling the parser to make more fine-grained distinctions, which better capture real
 5567 linguistic phenomena. However, each production is more rare, and so careful smoothing
 5568 or regularization is required to control the variance over production scores.

5569 10.5.2 Lexicalized context-free grammars

5570 The examples in § 10.2.2 demonstrate the importance of individual words in resolving
 5571 parsing ambiguity: the preposition *on* is more likely to attach to *met*, while the preposition
 5572 *of* is more likely to attachment to *President*. But of all word pairs, which are relevant to
 5573 attachment decisions? Consider the following variants on the original examples:

- 5574 (10.4) We met the President of Mexico.
- 5575 (10.5) We met the first female President of Mexico.
- 5576 (10.6) They had supposedly met the President on Monday.

5577 The underlined words are the **head words** of their respective phrases: *met* heads the verb
 5578 phrase, and *President* heads the direct object noun phrase. These heads provide useful
 5579 semantic information. But they break the context-free assumption, which states that the
 5580 score for a production depends only on the parent and its immediate children, and not
 5581 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,

$$\text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(of) \quad [10.38]$$

$$\text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(on). \quad [10.39]$$

5582 Lexicalization was a major step towards accurate PCFG parsing. It requires solving three
 5583 problems: identifying the heads of all constituents in a treebank; parsing efficiently while
 5584 keeping track of the heads; and estimating the scores for lexicalized productions.

5585 10.5.2.1 Identifying head words

5586 The head of a constituent is the word that is the most useful for determining how that
 5587 constituent is integrated into the rest of the sentence.⁵ The head word of a constituent is
 5588 determined recursively: for any non-terminal production, the head of the left-hand side
 5589 must be the head of one of the children. The head is typically selected according to a set of
 5590 deterministic rules, sometimes called **head percolation rules**. In many cases, these rules
 5591 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).

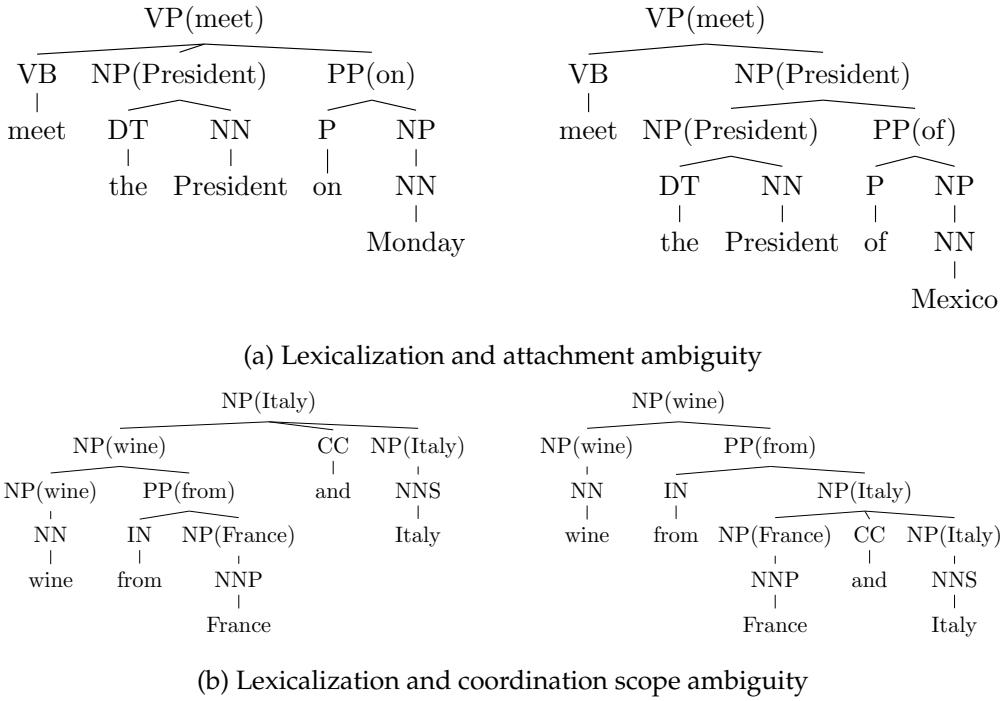


Figure 10.6: Examples of lexicalization

5592 of the noun; the head of a sentence in a $S \rightarrow NP VP$ production is the head of the verb
 5593 phrase.

5594 Table 10.3 shows a fragment of the head percolation rules used in many English pars-
 5595 ing systems. The meaning of the first rule is that to find the head of an S constituent, first
 5596 look for the rightmost VP child; if you don't find one, then look for the rightmost $SBAR$
 5597 child, and so on down the list. Verb phrases are headed by left verbs (the head of *can plan*
 5598 *on walking* is *planned*, since the modal verb *can* is tagged *MD*); noun phrases are headed by
 5599 the rightmost noun-like non-terminal (so the head of *the red cat* is *cat*),⁶ and prepositional
 5600 phrases are headed by the preposition (the head of *at Georgia Tech* is *at*). Some of these
 5601 rules are somewhat arbitrary — there's no particular reason why the head of *cats and dogs*
 5602 should be *dogs* — but the point here is just to get some lexical information that can support
 5603 parsing, not to make deep claims about syntax. Figure 10.6 shows the application of these
 5604 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>

5605 10.5.2.2 Parsing lexicalized context-free grammars

5606 A naïve application of lexicalization would simply increase the set of non-terminals by
 5607 taking the cross-product with the set of terminal symbols, so that the non-terminals now
 5608 include symbols like $NP(President)$ and $VP(meet)$. Under this approach, the CKY parsing
 5609 algorithm could be applied directly to the lexicalized production rules. However, the
 5610 complexity would be cubic in the size of the vocabulary of terminal symbols, which would
 5611 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:

$$t_\ell[i, j, h, X] = \max_{(X \rightarrow Y Z)} \max_{k > h} \max_{k < h' \leq j} t[i, k, h, Y] + t[k, j, h', Z] + \psi(X(h) \rightarrow Y(h)Z(h')) \quad [10.40]$$

$$t_r[i, j, h, X] = \max_{(X \rightarrow Y Z)} \max_{k < h} \max_{i < h' \leq k} t[i, k, h', Y] + t[k, j, h, Z] + (\psi(X(h) \rightarrow Y(h')Z(h))) \quad [10.41]$$

$$t[i, j, h, X] = \max(t_\ell[i, j, h, X], t_r[i, j, h, X]). \quad [10.42]$$

5612 To compute t_ℓ , we maximize over all split points $k > h$, since the head word must be in
 5613 the left child. We then maximize again over possible head words h' for the right child. An
 5614 analogous computation is performed for t_r . The size of the table is now $\mathcal{O}(M^3N)$, where
 5615 M is the length of the input and N is the number of non-terminals. Furthermore, each
 5616 cell is computed by performing $\mathcal{O}(M^2)$ operations, since we maximize over both the split
 5617 point k and the head h' . The time complexity of the algorithm is therefore $\mathcal{O}(RM^5N)$,
 5618 where R is the number of rules in the grammar. Fortunately, more efficient solutions are
 5619 possible. In general, the complexity of parsing can be reduced to $\mathcal{O}(M^4)$ in the length of

5620 the input; for a broad class of lexicalized CFGs, the complexity can be made cubic in the
 5621 length of the input, just as in unlexicalized CFGs (Eisner, 2000).

5622 10.5.2.3 Estimating lexicalized context-free grammars

5623 The final problem for lexicalized parsing is how to estimate weights for lexicalized pro-
 5624 ductions $X(i) \rightarrow Y(j) Z(k)$. These productions are said to be bilexical, because they
 5625 involve scores over pairs of words: in the example *meet the President of Mexico*, we hope
 5626 to choose the correct attachment point by modeling the bilexical affinities of (*meet, of*) and
 5627 (*President, of*). The number of such word pairs is quadratic in the size of the vocabulary,
 5628 making it difficult to estimate the weights of lexicalized production rules directly from
 5629 data. This is especially true for probabilistic context-free grammars, in which the weights
 5630 are obtained from smoothed relative frequency. In a treebank with a million tokens, a
 5631 vanishingly small fraction of the possible lexicalized productions will be observed more
 5632 than once.⁷ The Charniak (1997) and Collins (1997) parsers therefore focus on approxi-
 5633 mating the probabilities of lexicalized productions, using various smoothing techniques
 5634 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:

$$\begin{aligned} f(\text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(of)) = & \{\text{NP}(*) \rightarrow \text{NP}(*) \text{ PP}(*), \\ & \text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(*), \\ & \text{NP}(*) \rightarrow \text{NP}(*) \text{ PP}(of), \\ & \text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(of)\} \end{aligned}$$

5635 The first feature scores the unlexicalized production $\text{NP} \rightarrow \text{NP PP}$; the next two features
 5636 lexicalize only one element of the production, thereby scoring the appropriateness of NP
 5637 attachment for the individual words *President* and *of*; the final feature scores the specific
 5638 bilexical affinity of *President* and *of*. For bilexical pairs that are encountered frequently in
 5639 the treebank, this bilexical feature can play an important role in parsing; for pairs that are
 5640 absent or rare, regularization will drive its weight to zero, forcing the parser to rely on the
 5641 more coarse-grained features.

5642 In chapter 14, we will encounter techniques for clustering words based on their **distribu-**
 5643 **tional** properties — the contexts in which they appear. Such a clustering would group
 5644 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).

5645 as features in discriminative lexicalized parsing, striking a middle ground between full
 5646 lexicalization and non-terminals (Finkel et al., 2008). In this way, labeled examples con-
 5647 taining relatively common words like *whale* can help to improve parsing for rare words
 5648 like *beluga*, as long as those two words are clustered together.

5649 **10.5.3 *Refinement grammars**

5650 Lexicalization improves on context-free parsing by adding detailed information in the
 5651 form of lexical heads. However, estimating the scores of lexicalized productions is dif-
 5652 ficult. Klein and Manning (2003) argue that the right level of linguistic detail is some-
 5653 where between treebank categories and individual words. Some parts-of-speech and non-
 5654 terminals are truly substitutable: for example, *cat*/N and *dog*/N. But others are not: for
 5655 example, the preposition *of* exclusively attaches to nouns, while the preposition *as* is more
 5656 likely to modify verb phrases. Klein and Manning (2003) obtained a 2% improvement in
 5657 *F*-MEASURE on a parent-annotated PCFG parser by making a single change: splitting the
 5658 preposition category into six subtypes. They propose a series of linguistically-motivated
 5659 refinements to the Penn Treebank annotations, which in total yielded a 40% error reduc-
 5660 tion.

5661 Non-terminal refinement process can be automated by treating the refined categories
 5662 as latent variables. For example, we might split the noun phrase non-terminal into NP1, NP2, NP3, ...,
 5663 without defining in advance what each refined non-terminal corresponds to. This can
 5664 be treated as **partially supervised learning**, similar to the multi-component document
 5665 classification model described in § 5.2.3. A latent variable PCFG can be estimated by
 5666 expectation-maximization (Matsuzaki et al., 2005):

- 5667 • In the E-step, estimate a marginal distribution q over the refinement type of each
 5668 non-terminal in each derivation. These marginals are constrained by the original
 5669 annotation: an NP can be reannotated as NP4, but not as VP3. Marginal probabili-
 5670 ties over refined productions can be computed from the **inside-outside algorithm**,
 5671 as described in § 10.4.3, where the E-step enforces the constraints imposed by the
 5672 original annotations.
- 5673 • In the M-step, recompute the parameters of the grammar, by summing over the
 5674 probabilities of anchored productions that were computed in the E-step:

$$E[\text{count}(X \rightarrow Y Z)] = \sum_{i=0}^M \sum_{j=i}^M \sum_{k=i}^j p(X \rightarrow Y Z, (i, j, k) | \mathbf{w}). \quad [10.43]$$

5675 As usual, this process can be iterated to convergence. To determine the number of re-
 5676 finement types for each tag, Petrov et al. (2006) apply a split-merge heuristic; Liang et al.
 5677 (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).

5678 Some examples of refined non-terminals are shown in Table 10.4. The proper nouns
 5679 differentiate months, first names, middle initials, last names, first names of places, and
 5680 second names of places; each of these will tend to appear in different parts of grammatical
 5681 productions. The personal pronouns differentiate grammatical role, with PRP-0 appear-
 5682 ing in subject position at the beginning of the sentence (note the capitalization), PRP-1
 5683 appearing in subject position but not at the beginning of the sentence, and PRP-2 appear-
 5684 ing in object position.

5685 10.6 Beyond context-free parsing

5686 In the context-free setting, the score for a parse is a combination of the scores of individual
 5687 productions. As we have seen, these models can be improved by using finer-grained non-
 5688 terminals, via parent-annotation, lexicalization, and automated refinement. However, the
 5689 inherent limitations to the expressiveness of context-free parsing motivate the consider-
 5690 ation of other search strategies. These strategies abandon the optimality guaranteed by
 5691 bottom-up parsing, in exchange for the freedom to consider arbitrary properties of the
 5692 proposed parses.

5693 10.6.1 Reranking

5694 A simple way to relax the restrictions of context-free parsing is to perform a two-stage pro-
 5695 cess, in which a context-free parser generates a k -best list of candidates, and a **reranker**
 5696 then selects the best parse from this list (Charniak and Johnson, 2005; Collins and Koo,
 5697 2005). The reranker can be trained from an objective that is similar to multi-class classi-
 5698 fication: the goal is to learn weights that assign a high score to the reference parse, or to

5699 the parse on the k -best list that has the lowest error. In either case, the reranker need only
 5700 evaluate the K best parses, and so no context-free assumptions are necessary. This opens
 5701 the door to more expressive scoring functions:

- 5702 • It is possible to incorporate arbitrary non-local features, such as the structural par-
 5703 allelism and right-branching orientation of the parse (Charniak and Johnson, 2005).
 5704 • Reranking enables the use of **recursive neural networks**, in which each constituent
 5705 span $w_{i+1:j}$ receives a vector $\mathbf{u}_{i,j}$ which is computed from the vector representa-
 5706 tions of its children, using a composition function that is linked to the production
 5707 rule (Socher et al., 2013), e.g.,

$$\mathbf{u}_{i,j} = f \left(\Theta_{X \rightarrow Y} Z \begin{bmatrix} \mathbf{u}_{i,k} \\ \mathbf{u}_{k,j} \end{bmatrix} \right) \quad [10.44]$$

5708 The overall score of the parse can then be computed from the final vector, $\Psi(\tau) =$
 5709 $\theta \mathbf{u}_{0,M}$.

5710 Reranking can yield substantial improvements in accuracy. The main limitation is that it
 5711 can only find the best parse among the K -best offered by the generator, so it is inherently
 5712 limited by the ability of the bottom-up parser to find high-quality candidates.

5713 10.6.2 Transition-based parsing

5714 Structure prediction can be viewed as a form of search. An alternative to bottom-up pars-
 5715 ing is to read the input from left-to-right, gradually building up a parse structure through
 5716 a series of **transitions**. Transition-based parsing is described in more detail in the next
 5717 chapter, in the context of dependency parsing. However, it can also be applied to CFG
 5718 parsing, as briefly described here.

5719 For any context-free grammar, there is an equivalent **pushdown automaton**, a model
 5720 of computation that accepts exactly those strings that can be derived from the grammar.
 5721 This computational model consumes the input from left to right, while pushing and pop-
 5722 ping elements on a stack. This architecture provides a natural transition-based parsing
 5723 framework for context-free grammars, known as **shift-reduce parsing**.

5724 Shift-reduce parsing is a type of transition-based parsing, in which the parser can take
 5725 the following actions:

- 5726 • *shift* the next terminal symbol onto the stack;
 5727 • *unary-reduce* the top item on the stack, using a unary production rule in the gram-
 5728 mar;
 5729 • *binary-reduce* the top two items onto the stack, using a binary production rule in the
 5730 grammar.

5731 The set of available actions is constrained by the situation: the parser can only shift if
 5732 there are remaining terminal symbols in the input, and it can only reduce if an applicable
 5733 production rule exists in the grammar. If the parser arrives at a state where the input
 5734 has been completely consumed, and the stack contains only the element S, then the input
 5735 is accepted. If the parser arrives at a non-accepting state where there are no possible
 5736 actions, the input is rejected. A parse error occurs if there is some action sequence that
 5737 would accept an input, but the parser does not find it.

5738 **Example** Consider the input *we eat sushi* and the grammar in Table 10.1. The input can
 5739 be parsed through the following sequence of actions:

- 5740 1. **Shift** the first token *we* onto the stack.
- 5741 2. **Reduce** the top item on the stack to NP, using the production $NP \rightarrow we$.
- 5742 3. **Shift** the next token *eat* onto the stack, and **reduce** it to V with the production $V \rightarrow eat$.
- 5744 4. **Shift** the final token *sushi* onto the stack, and **reduce** it to NP. The input has been
 5745 completely consumed, and the stack contains [NP, V, NP].
- 5746 5. **Reduce** the top two items using the production $VP \rightarrow V NP$. The stack now con-
 5747 tains [VP, NP].
- 5748 6. **Reduce** the top two items using the production $S \rightarrow NP VP$. The stack now contains
 5749 [S]. Since the input is empty, this is an accepting state.

5750 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).
 5755 This is far better than the cubic time complexity required by CKY parsing.

5756 **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 .
 5759 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

5766 independence assumptions required by chart parsing; search errors are the price that must
 5767 be paid for this flexibility.

5768 **Learning transition-based parsing** Transition-based parsing can be combined with ma-
 5769 chine learning by training a classifier to select the correct action in each situation. This
 5770 classifier is free to choose any feature of the input, the state of the parser, and the parse
 5771 history. However, there is no optimality guarantee: the parser may choose a suboptimal
 5772 parse, due to a mistake at the beginning of the analysis. Nonetheless, some of the strongest
 5773 CFG parsers are based on the shift-reduce architecture, rather than CKY. A recent genera-
 5774 tion of models links shift-reduce parsing with recurrent neural networks, updating a
 5775 hidden state vector while consuming the input (e.g., Cross and Huang, 2016; Dyer et al.,
 5776 2016). Learning algorithms for transition-based parsing are discussed in more detail in
 5777 § 11.3.

5778 Exercises

5779 1. Design a grammar that handles English subject-verb agreement. Specifically, your
 5780 grammar should handle the examples below correctly:

5781 (10.7) a. She sings.

5782 b. We sing.

5783 (10.8) a. *She sing.

5784 b. *We sings.

5785 2. Extend your grammar from the previous problem to include the auxiliary verb *can*,
 5786 so that the following cases are handled:

5787 (10.9) a. She can sing.

5788 b. We can sing.

5789 (10.10) a. *She can sings.

5790 b. *We can sings.

5791 3. French requires subjects and verbs to agree in person and number, and it requires
 5792 determiners and nouns to agree in gender and number. Verbs and their objects need
 5793 not agree. Assuming that French has two genders (feminine and masculine), three
 5794 persons (first [*me*], second [*you*], third [*her*]), and two numbers (singular and plural),
 5795 how many productions are required to extend the following simple grammar to
 5796 handle agreement?

5797 $S \rightarrow NP\ VP$
 $VP \rightarrow V \mid V\ NP \mid V\ NP\ NP$
 $NP \rightarrow DET\ NN$

5798 4. Consider the grammar:

5799 $S \rightarrow NP\ VP$
 $VP \rightarrow V\ NP$
 $NP \rightarrow JJ\ NP$
 $NP \rightarrow fish\ (\text{the animal})$
 $V \rightarrow fish\ (\text{the action of fishing})$
 $JJ \rightarrow fish\ (\text{a modifier, as in } fish\ sauce \text{ or } fish\ stew)$

5800 Apply the CKY algorithm and identify all possible parses for the sentence *fish fish fish fish*.

5801

5802 5. Choose one of the possible parses for the previous problem, and show how it can be
 derived by a series of shift-reduce actions.

5804 6. To handle VP coordination, a grammar includes the production $VP \rightarrow VP\ CC\ VP$.
 5805 To handle adverbs, it also includes the production $VP \rightarrow VP\ ADV$. Assume all verbs
 5806 are generated from a sequence of unary productions, e.g., $VP \rightarrow V \rightarrow eat$.

- 5807 a) Show how to binarize the production $VP \rightarrow VP\ CC\ VP$.
- 5808 b) Use your binarized grammar to parse the sentence *They eat and drink together*,
 5809 treating *together* as an adverb.
- 5810 c) Prove that a weighted CFG cannot distinguish the two possible derivations of
 5811 this sentence. Your explanation should focus on the productions in the original,
 5812 non-binary grammar.
- 5813 d) Explain what condition must hold for a parent-annotated WCFG to prefer the
 5814 derivation in which *together* modifies the coordination *eat and drink*.

5815 7. Consider the following PCFG:
 5816

$$p(X \rightarrow X\ X) = \frac{1}{2} \quad [10.45]$$

$$p(X \rightarrow Y) = \frac{1}{2} \quad [10.46]$$

$$p(Y \rightarrow \sigma) = \frac{1}{|\Sigma|}, \forall \sigma \in \Sigma \quad [10.47]$$

- 5815 a) Compute the probability $p(\hat{\tau})$ of the maximum probability parse for a string
 5816 $w \in \Sigma^M$.

- 5817 b) Compute the conditional probability $p(\hat{\tau} \mid \mathbf{w})$.
- 5818 8. Context-free grammars can be used to parse the internal structure of words. Us-
 5819 ing the weighted CKY algorithm and the following weighted context-free grammar,
 5820 identify the best parse for the sequence of morphological segments *in+flame+able*.
-
- | | | | |
|---------|---|--------------|----|
| S | → | V | 0 |
| S | → | N | 0 |
| S | → | J | 0 |
| V | → | VPref N | -1 |
| J | → | N JSuff | 1 |
| J | → | V JSuff | 0 |
| J | → | NegPref J | 1 |
| VPref | → | <i>in+</i> | 2 |
| NegPref | → | <i>in+</i> | 1 |
| N | → | <i>flame</i> | 0 |
| JSuff | → | <i>+able</i> | 0 |
- 5821
- 5822 9. Use the inside and outside scores to compute the marginal probability $p(X_{i:j} \rightarrow Y_{i:k-1} Z_{k:j} \mid \mathbf{w})$,
 5823 indicating that Y spans $\mathbf{w}_{i:k-1}$, Z spans $\mathbf{w}_{k:j}$, and X is the parent of Y and Z , span-
 5824 ning $\mathbf{w}_{i:j}$.
- 5825 10. Suppose that the potentials $\Psi(X \rightarrow \alpha)$ are log-probabilities, so that $\sum_{\alpha} \exp \Psi(X \rightarrow \alpha) = 1$
 5826 for all X . Verify that the semiring inside recurrence from Equation 10.26 generates
 5827 the log-probability $\log p(\mathbf{w}) = \log \sum_{\tau: \text{yield}(\tau)=\mathbf{w}} p(\tau)$.

5828 Chapter 11

5829 Dependency parsing

5830 The previous chapter discussed algorithms for analyzing sentences in terms of nested con-
5831 stituents, such as noun phrases and verb phrases. However, many of the key sources of
5832 ambiguity in phrase-structure analysis relate to questions of **attachment**: where to attach a
5833 prepositional phrase or complement clause, how to scope a coordinating conjunction, and
5834 so on. These attachment decisions can be represented with a more lightweight structure:
5835 a directed graph over the words in the sentence, known as a **dependency parse**. Syntac-
5836 tic annotation has shifted its focus to such dependency structures: at the time of this
5837 writing, the **Universal Dependencies** project offers more than 100 dependency treebanks
5838 for more than 60 languages.¹ This chapter will describe the linguistic ideas underlying
5839 dependency grammar, and then discuss exact and transition-based parsing algorithms.
5840 The chapter will also discuss recent research on **learning to search** in transition-based
5841 structure prediction.

5842 11.1 Dependency grammar

5843 While **dependency grammar** has a rich history of its own (Tesnière, 1966; Kübler et al.,
5844 2009), it can be motivated by extension from the lexicalized context-free grammars that
5845 we encountered in previous chapter (§ 10.5.2). Recall that lexicalization augments each
5846 non-terminal with a **head word**. The head of a constituent is identified recursively, using
5847 a set of **head rules**, as shown in Table 10.3. An example of a lexicalized context-free parse
5848 is shown in Figure 11.1a. In this sentence, the head of the S constituent is the main verb,
5849 *scratch*; this non-terminal then produces the noun phrase *the cats*, whose head word is
5850 *cats*, and from which we finally derive the word *the*. Thus, the word *scratch* occupies the
5851 central position for the sentence, with the word *cats* playing a supporting role. In turn, *cats*

¹universaldependencies.org

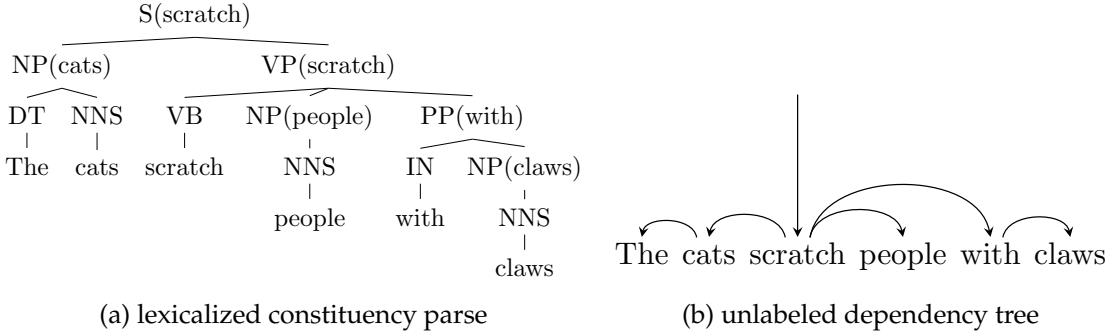


Figure 11.1: Dependency grammar is closely linked to lexicalized context free grammars: each lexical head has a dependency path to every other word in the constituent. (This example is based on the lexicalization rules from § 10.5.2, which make the preposition the head of a prepositional phrase. In the more contemporary Universal Dependencies annotations, the head of *with claws* would be *claws*, so there would be an edge *scratch* → *claws*.)

5852 occupies the central position for the noun phrase, with the word *the* playing a supporting
5853 role.

5854 The relationships between words in a sentence can be formalized in a directed graph,
5855 based on the lexicalized phrase-structure parse: create an edge (i, j) iff word i is the head
5856 of a phrase whose child is a phrase headed by word j . Thus, in our example, we would
5857 have *scratch* → *cats* and *cats* → *the*. We would not have the edge *scratch* → *the*, because
5858 although $S(\text{scratch})$ dominates $\text{DET}(\text{the})$ in the phrase-structure parse tree, it is not its im-
5859 mediate parent. These edges describe **syntactic dependencies**, a bilexical relationship
5860 between a **head** and a **dependent**, which is at the heart of dependency grammar.

5861 Continuing to build out this **dependency graph**, we will eventually reach every word
5862 in the sentence, as shown in Figure 11.1b. In this graph — and in all graphs constructed
5863 in this way — every word has exactly one incoming edge, except for the root word, which
5864 is indicated by a special incoming arrow from above. Furthermore, the graph is *weakly*
5865 *connected*: if the directed edges were replaced with undirected edges, there would be a
5866 path between all pairs of nodes. From these properties, it can be shown that there are no
5867 cycles in the graph (or else at least one node would have to have more than one incoming
5868 edge), and therefore, the graph is a tree. Because the graph includes all vertices, it is a
5869 **spanning tree**.

5870 11.1.1 Heads and dependents

5871 A dependency edge implies an asymmetric syntactic relationship between the head and
5872 dependent words, sometimes called **modifiers**. For a pair like *the cats* or *cats scratch*, how

5873 do we decide which is the head? Here are some possible criteria:

- 5874 • 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.
- 5876 • The modifier may be optional while the head is mandatory: for example, in the
5877 sentence *cats scratch people with claws*, the subtrees *cats scratch* and *cats scratch people*
5878 are grammatical sentences, but *with claws* is not.
- 5879 • The head determines the morphological form of the modifier: for example, in lan-
5880 guages that require gender agreement, the gender of the noun determines the gen-
5881 der of the adjectives and determiners.
- 5882 • Edges should first connect content words, and then connect function words.

5883 As always, these guidelines sometimes conflict. The Universal Dependencies (UD)
5884 project has attempted to identify a set of principles that can be applied to dozens of dif-
5885 ferent languages (Nivre et al., 2016).² These guidelines are based on the universal part-
5886 of-speech tags from chapter 8. They differ somewhat from the head rules described in
5887 § 10.5.2: for example, on the principle that dependencies should relate content words, the
5888 prepositional phrase *with claws* would be headed by *claws*, resulting in an edge *scratch* →
5889 *claws*, and another edge *claws* → *with*.

5890 One objection to dependency grammar is that not all syntactic relations are asymmet-
5891 ric. Coordination is one of the most obvious examples (Popel et al., 2013): in the sentence,
5892 *Abigail and Max like kimchi* (Figure 11.2), which word is the head of the coordinated noun
5893 phrase *Abigail and Max*? Choosing either *Abigail* or *Max* seems arbitrary; fairness argues
5894 for making *and* the head, but this seems like the least important word in the noun phrase,
5895 and selecting it would violate the principle of linking content words first. The Universal
5896 Dependencies annotation system arbitrarily chooses the left-most item as the head — in
5897 this case, *Abigail* — and includes edges from this head to both *Max* and the coordinating
5898 conjunction *and*. These edges are distinguished by the labels CONJ (for the thing begin
5899 conjoined) and CC (for the coordinating conjunction). The labeling system is discussed
5900 next.

5901 11.1.2 Labeled dependencies

5902 Edges may be **labeled** to indicate the nature of the syntactic relation that holds between
5903 the two elements. For example, in Figure 11.2, the label NSUBJ on the edge from *like* to
5904 *Abigail* indicates that the subtree headed by *Abigail* is the noun subject of the verb *like*;
5905 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

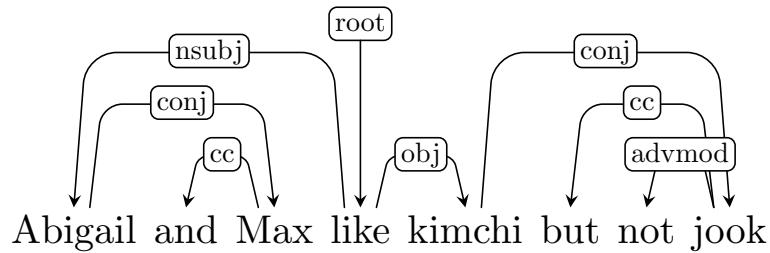


Figure 11.2: In the Universal Dependencies annotation system, the left-most item of a coordination is the head.

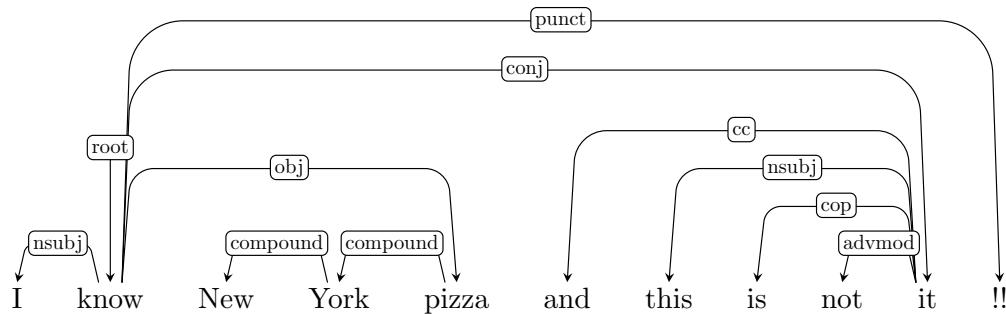


Figure 11.3: A labeled dependency parse from the English UD Treebank (reviews-361348-0006)

5906 *kimchi* is the object.³ The negation *not* is treated as an adverbial modifier (ADVMOD) on
5907 the noun *jook*.

5908 A slightly more complex example is shown in Figure 11.3. The multiword expression
5909 *New York pizza* is treated as a “flat” unit of text, with the elements linked by the COM-
5910 POUND relation. The sentence includes two clauses that are conjoined in the same way
5911 that noun phrases are conjoined in Figure 11.2. The second clause contains a **copula** verb
5912 (see § 8.1.1). For such clauses, we treat the “object” of the verb as the root — in this case,
5913 *it* — and label the verb as a dependent, with the COP relation. This example also shows
5914 how punctuations are treated, with label PUNCT.

5915 11.1.3 Dependency subtrees and constituents

5916 Dependency trees hide information that would be present in a CFG parse. Often what
5917 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.

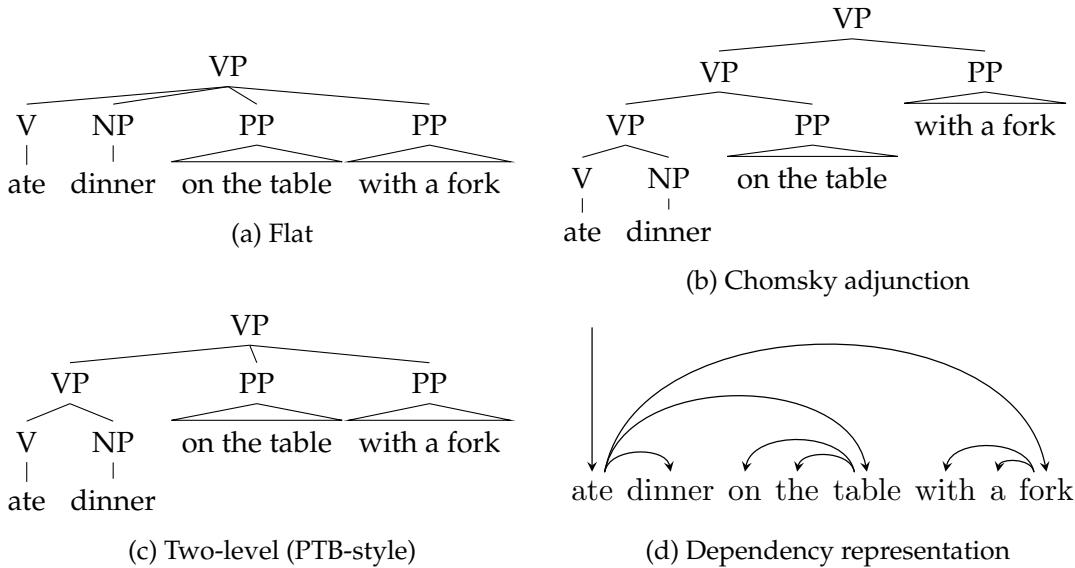


Figure 11.4: The three different CFG analyses of this verb phrase all correspond to a single dependency structure.

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)

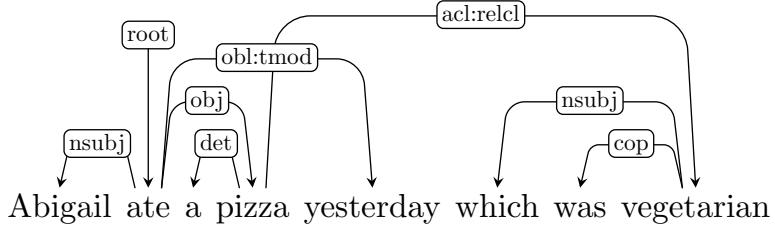


Figure 11.5: An example of a non-projective dependency parse. The “crossing edge” arises from the relative clause *which was vegetarian* and the oblique temporal modifier *yesterday*.

5937 **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.*

5939 Figure 11.5 gives an example of a non-projective dependency graph in English. This
 5940 dependency graph does not correspond to any constituent parse. As shown in Table 11.1,
 5941 non-projectivity is more common in languages such as Czech and German. Even though
 5942 relatively few dependencies are non-projective in these languages, many sentences have
 5943 at least one such dependency. As we will soon see, projectivity has important algorithmic
 5944 consequences.

5945 11.2 Graph-based dependency parsing

5946 Let $\mathbf{y} = \{i \xrightarrow{r} j\}$ represent a dependency graph, in which each edge is a relation r from
 5947 head word $i \in \{1, 2, \dots, M, \text{ROOT}\}$ to modifier $j \in \{1, 2, \dots, M\}$. The special node ROOT
 5948 indicates the root of the graph, and M is the length of the input $|\mathbf{w}|$. Given a scoring
 5949 function $\Psi(\mathbf{y}, \mathbf{w}; \theta)$, the optimal parse is,

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}(\mathbf{w})}{\operatorname{argmax}} \Psi(\mathbf{y}, \mathbf{w}; \theta), \quad [11.1]$$

5950 where $\mathcal{Y}(\mathbf{w})$ is the set of valid dependency parses on the input \mathbf{w} . As usual, the number
 5951 of possible labels $|\mathcal{Y}(\mathbf{w})|$ is exponential in the length of the input (Wu and Chao, 2004).

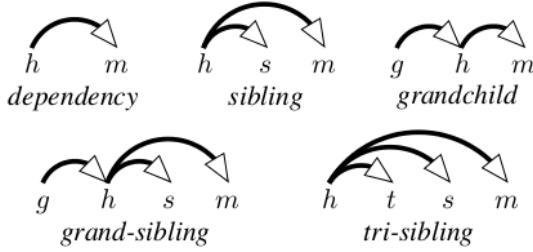


Figure 11.6: Feature templates for higher-order dependency parsing (Koo and Collins, 2010) [todo: permission]

5952 Algorithms that search over this space of possible graphs are known as **graph-based de-**
5953 **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 $i \rightarrow j$:

$$\Psi(\mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}). \quad [11.2]$$

5954 Dependency parsers that operate under this assumption are known as **arc-factored**, since
5955 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,

$$\begin{aligned} \Psi(\mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) &= \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi_{\text{parent}}(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) \\ &\quad + \sum_{k \xrightarrow{r'} i \in \mathbf{y}} \psi_{\text{grandparent}}(i \xrightarrow{r} j, k, r', \mathbf{w}; \boldsymbol{\theta}) \end{aligned} \quad [11.3]$$

$$+ \sum_{\substack{i \xrightarrow{r'} s \in \mathbf{y} \\ s \neq j}} \psi_{\text{sibling}}(i \xrightarrow{r} j, s, r', \mathbf{w}; \boldsymbol{\theta}). \quad [11.4]$$

5956 The top line scores computes a scoring function that includes the grandparent *k*; the
5957 bottom line computes a scoring function for each sibling *s*. For projective dependency

graphs, there are efficient algorithms for second-order and third-order dependency parsing (Eisner, 1996; McDonald and Pereira, 2006; Koo and Collins, 2010); for non-projective dependency graphs, second-order dependency parsing is NP-hard (McDonald and Pereira, 2006). The specific algorithms are discussed in the next section.

11.2.1 Graph-based parsing algorithms

The distinction between projective and non-projective dependency trees (§ 11.1.3) plays a key role in the choice of algorithms. Because projective dependency trees are closely related to (and can be derived from) lexicalized constituent trees, lexicalized parsing algorithms can be applied directly. For the more general problem of parsing to arbitrary spanning trees, a different class of algorithms is required. In both cases, arc-factored dependency parsing relies on precomputing the scores $\psi(i \xrightarrow{r} j, w; \theta)$ for each potential edge. There are $\mathcal{O}(M^2 R)$ such scores, where M is the length of the input and R is the number of dependency relation types, and this is a lower bound on the time and space complexity of any exact algorithm for arc-factored dependency parsing.

11.2.1.1 Projective dependency parsing

Any lexicalized constituency tree can be converted into a projective dependency tree by creating arcs between the heads of constituents and their parents, so any algorithm for lexicalized constituent parsing can be converted into an algorithm for projective dependency parsing, by converting arc scores into scores for lexicalized productions. As noted in § 10.5.2, there are cubic time algorithms for lexicalized constituent parsing, which are extensions of the CKY algorithm. Therefore, arc-factored projective dependency parsing can be performed in cubic time in the length of the input.

Second-order projective dependency parsing can also be performed in cubic time, with minimal modifications to the lexicalized parsing algorithm (Eisner, 1996). It is possible to go even further, to **third-order dependency parsing**, in which the scoring function may consider great-grandparents, grand-siblings, and “tri-siblings”, as shown in Figure 11.6. Third-order dependency parsing can be performed in $\mathcal{O}(M^4)$ time, which can be made practical through the use of pruning to eliminate unlikely edges (Koo and Collins, 2010).

11.2.1.2 Non-projective dependency parsing

In non-projective dependency parsing, the goal is to identify the highest-scoring spanning tree over the words in the sentence. The arc-factored assumption ensures that the score for each spanning tree will be computed as a sum over scores for the edges, which are precomputed. Based on these scores, we build a weighted connected graph. Arc-factored non-projective dependency parsing is then equivalent to finding the spanning 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-**

5993 **Liu-Edmonds algorithm** (Chu and Liu, 1965; Edmonds, 1967) computes this **maximum**
 5994 **spanning tree** efficiently. It does this by first identifying the best incoming edge $i \xrightarrow{r} j$ for
 5995 each vertex j . If the resulting graph does not contain cycles, it is the maximum spanning
 5996 tree. If there is a cycle, it is collapsed into a super-vertex, whose incoming and outgoing
 5997 edges are based on the edges to the vertices in the cycle. The algorithm is then applied
 5998 recursively to the resulting graph, and process repeats until a graph without cycles is
 5999 obtained.

6000 The time complexity of identifying the best incoming edge for each vertex is $\mathcal{O}(M^2R)$,
 6001 where M is the length of the input and R is the number of relations; in the worst case, the
 6002 number of cycles is $\mathcal{O}(M)$. Therefore, the complexity of the Chu-Liu-Edmonds algorithm
 6003 is $\mathcal{O}(M^3R)$. This complexity can be reduced to $\mathcal{O}(M^2N)$ by storing the edge scores in a
 6004 Fibonacci heap (Gabow et al., 1986). For more detail on graph-based parsing algorithms,
 6005 see Eisner (1997) and Kübler et al. (2009).

6006 **Higher-order non-projective dependency parsing** Given the tractability of higher-order
 6007 projective dependency parsing, you may be surprised to learn that non-projective second-
 6008 order dependency parsing is NP-Hard. This can be proved by reduction from the vertex
 6009 cover problem (Neuhaus and Bröker, 1997). A heuristic solution is to do projective pars-
 6010 ing first, and then post-process the projective dependency parse to add non-projective
 6011 edges (Nivre and Nilsson, 2005). More recent work has applied techniques for approxi-
 6012 mate inference in graphical models, including belief propagation (Smith and Eisner, 2008),
 6013 integer linear programming (Martins et al., 2009), variational inference (Martins et al.,
 6014 2010), and Markov Chain Monte Carlo (Zhang et al., 2014).

6015 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:

$$\text{Linear} \quad \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \boldsymbol{\theta} \cdot \mathbf{f}(i \xrightarrow{r} j, \mathbf{w}) \quad [11.5]$$

$$\text{Neural} \quad \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \text{Feedforward}([\mathbf{u}_{w_i}; \mathbf{u}_{w_j}]; \boldsymbol{\theta}) \quad [11.6]$$

$$\text{Generative} \quad \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \log p(w_j, r | w_i). \quad [11.7]$$

6016 11.2.2.1 Linear feature-based arc scores

6017 Linear models for dependency parsing incorporate many of the same features used in
 6018 sequence labeling and discriminative constituent parsing. These include:

- 6019 • the length and direction of the arc;
- 6020 • the words w_i and w_j linked by the dependency relation;
- 6021 • the prefixes, suffixes, and parts-of-speech of these words;

- 6022 • the neighbors of the dependency arc, $w_{i-1}, w_{i+1}, w_{j-1}, w_{j+1}$;
 6023 • the prefixes, suffixes, and part-of-speech of these neighbor words.

6024 Each of these features can be conjoined with the dependency edge label r . Note that
 6025 features in an arc-factored parser can refer to words other than w_i and w_j . The restriction
 6026 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:

$$\begin{aligned} \mathbf{f}(3 \rightarrow 5, \text{we eat sushi with chopsticks}) = & \langle \text{sushi} \rightarrow \text{chopsticks}, \\ & \text{sushi} \rightarrow \text{NNS}, \\ & \text{NN} \rightarrow \text{chopsticks}, \\ & \text{NNS} \rightarrow \text{NN} \rangle. \end{aligned}$$

6027 Regularized discriminative learning algorithms can then trade off between features at
 6028 varying levels of detail. McDonald et al. (2005) take this approach as far as *tetralexical*
 6029 features (e.g., $(w_i, w_{i+1}, w_{j-1}, w_j)$). Such features help to avoid choosing arcs that are un-
 6030 likely due to the intervening words: for example, there is unlikely to be an edge between
 6031 two nouns if the intervening span contains a verb. A large list of first and second-order
 6032 features is provided by Bochnet (2010), who uses a hashing function to store these features
 6033 efficiently.

6034 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:

$$\psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \text{FeedForward}([\mathbf{x}_i; \mathbf{x}_j]; \boldsymbol{\theta}_r), \quad [11.8]$$

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,

$$\mathbf{z} = g(\boldsymbol{\Theta}_r[\mathbf{x}_i; \mathbf{x}_j] + \mathbf{b}_r^{(z)}) \quad [11.9]$$

$$\psi(i \xrightarrow{r} j) = \boldsymbol{\beta}_r \mathbf{z} + \mathbf{b}_r^{(y)}, \quad [11.10]$$

6035 where $\boldsymbol{\Theta}_r$ is a matrix, $\boldsymbol{\beta}_r$ is a vector, each b_r is a scalar, and the function g is an elementwise
 6036 tanh activation function.

6037 The vector \mathbf{x}_i can be set equal to the word embedding, which may be pre-trained or
 6038 learned by backpropagation (Pei et al., 2015). Alternatively, contextual information can
 6039 be incorporated by applying a bidirectional recurrent neural network across the input, as
 6040 described in § 7.6. The RNN hidden states at each word can be used as inputs to the arc
 6041 scoring function (Kiperwasser and Goldberg, 2016).

6042 **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*,

$$\mathbf{y} = \{(ROOT, 2), (2, 1), (2, 3), (3, 5), (5, 4)\} \quad [11.11]$$

$$\log p(\mathbf{w} \mid \mathbf{y}) = \sum_{(i \rightarrow j) \in \mathbf{y}} \log p(w_j \mid w_i) \quad [11.12]$$

$$\begin{aligned} &= \log p(eat \mid ROOT) + \log p(we \mid eat) + \log p(sushi \mid eat) \\ &\quad + \log p(rice \mid sushi) + \log p(with \mid rice). \end{aligned} \quad [11.13]$$

6043 Probabilistic generative models are used in combination with expectation-maximization
 6044 (chapter 5) for unsupervised dependency parsing (Klein and Manning, 2004).

6045 **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,

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \mathbf{y}') \quad [11.14]$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \mathbf{f}(\mathbf{w}, \mathbf{y}) - \mathbf{f}(\mathbf{w}, \hat{\mathbf{y}}) \quad [11.15]$$

6046 In this case, the argmax requires a maximization over all dependency trees for the sen-
 6047 tence, which can be computed using the algorithms described in § 11.2.1. We can apply
 6048 all the usual tricks from § 2.2: weight averaging, a large margin objective, and regular-
 6049 ization. McDonald et al. (2005) were the first to treat dependency parsing as a structure
 6050 prediction problem, using MIRA, an online margin-based learning algorithm. Neural arc
 6051 scores can be learned in the same way, backpropagating from a margin loss to updates on
 6052 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,

$$p(\mathbf{y} \mid \mathbf{w}) = \frac{\exp \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})}{\sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp \sum_{i \xrightarrow{r} j \in \mathbf{y}'} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})} \quad [11.16]$$

Such a model is trained to minimize the negative log conditional-likelihood. Just as in CRF sequence models (§ 7.5.3) and the logistic regression classifier (§ 2.4), the gradients involve marginal probabilities $p(i \xrightarrow{r} j \mid \mathbf{w}; \boldsymbol{\theta})$, which in this case are probabilities over individual dependencies. In arc-factored models, these probabilities can be computed in polynomial time. For projective dependency trees, the marginal probabilities can be computed in cubic time, using a variant of the inside-outside algorithm (Lari and Young, 1990). For non-projective dependency parsing, marginals can also be computed in cubic time, using the **matrix-tree theorem** (Koo et al., 2007; McDonald et al., 2007; Smith and Smith, 2007). Details of these methods are described by Kübler et al. (2009).

11.3 Transition-based dependency parsing

Graph-based dependency parsing offers exact inference, meaning that it is possible to recover the best-scoring parse for any given model. But this comes at a price: the scoring function is required to decompose into local parts — in the case of non-projective parsing, these parts are restricted to individual arcs. These limitations are felt more keenly in dependency parsing than in sequence labeling, because second-order dependency features are critical to correctly identify some types of attachments. For example, prepositional phrase attachment depends on the attachment point, the object of the preposition, and the preposition itself; arc-factored scores cannot account for all three of these features simultaneously. Graph-based dependency parsing may also be criticized on the basis of intuitions about human language processing: people read and listen to sentences *sequentially*, incrementally building mental models of the sentence structure and meaning before getting to the end (Jurafsky, 1996). This seems hard to reconcile with graph-based algorithms, which perform bottom-up operations on the entire sentence, requiring the parser to keep every word in memory. Finally, from a practical perspective, graph-based dependency parsing is relatively slow, running in cubic time in the length of the input.

Transition-based algorithms address all three of these objections. They work by moving through the sentence sequentially, while performing actions that incrementally update a stored representation of what has been read thus far. As with the shift-reduce parser from § 10.6.2, this representation consists of a stack, onto which parsing substructures can be pushed and popped. In shift-reduce, these substructures were constituents; in the transition systems that follow, they will be projective dependency trees over partial

spans of the input.⁴ Parsing is complete when the input is consumed and there is only a single structure on the stack. The sequence of actions that led to the parse is known as the **derivation**. One problem with transition-based systems is that there may be multiple derivations for a single parse structure — a phenomenon known as **spurious ambiguity**.

11.3.1 Transition systems for dependency parsing

A **transition system** consists of a representation for describing configurations of the parser, and a set of transition actions, which manipulate the configuration. There are two main transition systems for dependency parsing: **arc-standard**, which is closely related to shift-reduce, and **arc-eager**, which adds an additional action that can simplify derivations (Abney and Johnson, 1991). In both cases, transitions are between **configurations** that are represented as triples, $C = (\sigma, \beta, A)$, where σ is the stack, β is the input buffer, and A is the list of arcs that have been created (Nivre, 2008). In the initial configuration,

$$C_{\text{initial}} = ([\text{ROOT}], \mathbf{w}, \emptyset), \quad [11.17]$$

indicating that the stack contains only the special node ROOT, the entire input is on the buffer, and the set of arcs is empty. An accepting configuration is,

$$C_{\text{accept}} = ([\text{ROOT}], \emptyset, A), \quad [11.18]$$

where the stack contains only ROOT, the buffer is empty, and the arcs A define a spanning tree over the input. The arc-standard and arc-eager systems define a set of transitions between configurations, which are capable of transforming an initial configuration into an accepting configuration. In both of these systems, the number of actions required to parse an input grows linearly in the length of the input, making transition-based parsing considerably more efficient than graph-based methods.

11.3.1.1 Arc-standard

The **arc-standard** transition system is closely related to shift-reduce, and to the LR algorithm that is used to parse programming languages (Aho et al., 2006). It includes the following classes of actions:

- SHIFT: move the first item from the input buffer on to the top of the stack,

$$(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A), \quad [11.19]$$

where we write $i|\beta$ to indicate that i is the leftmost item in the input buffer, and $\sigma|i$ to indicate the result of pushing i on to stack σ .

⁴Transition systems also exist for non-projective dependency parsing (e.g., Nivre, 2008).

- 6111 • ARC-LEFT: create a new left-facing arc of type r between the item on the top of the
 6112 stack and the first item in the input buffer. The head of this arc is j , which remains
 6113 at the front of the input buffer. The arc $j \xrightarrow{r} i$ is added to A . Formally,

$$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \oplus j \xrightarrow{r} i), \quad [11.20]$$

6114 where r is the label of the dependency arc, and \oplus concatenates the new arc $j \xrightarrow{r} i$ to
 6115 the list A .

- 6116 • ARC-RIGHT: creates a new right-facing arc of type r between the item on the top of the
 6117 stack and the first item in the input buffer. The head of this arc is i , which is
 6118 “popped” from the stack and pushed to the front of the input buffer. The arc $i \xrightarrow{r} j$
 6119 is added to A . Formally,

$$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, i|\beta, A \oplus i \xrightarrow{r} j), \quad [11.21]$$

6120 where again r is the label of the dependency arc.

6121 Each action has preconditions. The SHIFT action can be performed only when the buffer
 6122 has at least one element. The ARC-LEFT action cannot be performed when the root node
 6123 ROOT is on top of the stack, since this node must be the root of the entire tree. The ARC-
 6124 LEFT and ARC-RIGHT remove the modifier words from the stack (in the case of ARC-LEFT)
 6125 and from the buffer (in the case of ARC-RIGHT), so it is impossible for any word to have
 6126 more than one parent. Furthermore, the end state can only be reached when every word is
 6127 removed from the buffer and stack, so the set of arcs is guaranteed to constitute a spanning
 6128 tree. An example arc-standard derivation is shown in Table 11.2.

6129 11.3.1.2 Arc-eager dependency parsing

6130 In the arc-standard transition system, a word is completely removed from the parse once
 6131 it has been made the modifier in a dependency arc. At this time, any dependents of
 6132 this word must have already been identified. Right-branching structures are common in
 6133 English (and many other languages), with words often modified by units such as prepo-
 6134 sitional phrases to their right. In the arc-standard system, this means that we must first
 6135 shift all the units of the input onto the stack, and then work backwards, creating a series of
 6136 arcs, as occurs in Table 11.2. Note that the decision to shift *bagels* onto the stack guarantees
 6137 that the prepositional phrase *with lox* will attach to the noun phrase, and that this decision
 6138 must be made before the prepositional phrase is itself parsed. This has been argued to be
 6139 cognitively implausible (Abney and Johnson, 1991); from a computational perspective, it
 6140 means that a parser may need to look several steps ahead to make the correct decision.

6141 **Arc-eager dependency parsing** changes the ARC-RIGHT action so that right depen-
 6142 dents can be attached before all of their dependents have been found. Rather than re-
 6143 moving the modifier from both the buffer and stack, the ARC-RIGHT action pushes the

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*.

6144 modifier on to the stack, on top of the head. Because the stack can now contain elements
 6145 that already have parents in the partial dependency graph, two additional changes are
 6146 necessary:

- 6147 • A precondition is required to ensure that the ARC-LEFT action cannot be applied
 6148 when the top element on the stack already has a parent in A .
 6149 • A new REDUCE action is introduced, which can remove elements from the stack if
 6150 they already have a parent in A :

$$(\sigma|i, \beta, A) \Rightarrow (\sigma, \beta, A). \quad [11.22]$$

6151 As a result of these changes, it is now possible to create the arc *like* \rightarrow *bagels* before parsing
 6152 the prepositional phrase *with lox*. Furthermore, this action does not imply a decision about
 6153 whether the prepositional phrase will attach to the noun or verb. Noun attachment is
 6154 chosen in the parse in Table 11.3, but verb attachment could be achieved by applying the
 6155 REDUCE action at step 5 or 7.

6156 11.3.1.3 Projectivity

6157 The arc-standard and arc-eager transition systems are guaranteed to produce projective
 6158 dependency trees, because all arcs are between the word at the top of the stack and the
 6159 left-most edge of the buffer (Nivre, 2008). Non-projective transition systems can be con-
 6160 structed by adding actions that create arcs with words that are second or third in the
 6161 stack (Attardi, 2006), or by adopting an alternative configuration structure, which main-
 6162 tains a list of all words that do not yet have heads (Covington, 2001). In **pseudo-projective**

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*.

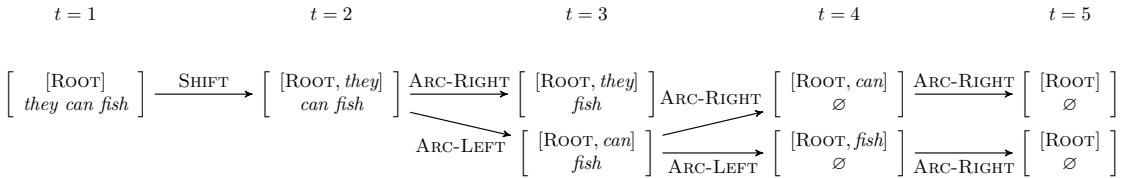


Figure 11.7: Beam search for unlabeled dependency parsing, with beam size $K = 2$. The arc lists for each configuration are not shown, but can be computed from the transitions.

6163 **dependency parsing**, a projective dependency parse is generated first, and then a set of
 6164 graph transformation techniques are applied, producing non-projective edges (Nivre and
 6165 Nilsson, 2005).

6166 11.3.1.4 Beam search

6167 In “greedy” transition-based parsing, the parser tries to make the best decision at each
 6168 configuration. This can lead to search errors, when an early decision locks the parser into
 6169 a poor derivation. For example, in Table 11.2, if ARC-RIGHT were chosen at step 4, then
 6170 the parser would later be forced to attach the prepositional phrase *with lox* to the verb
 6171 *likes*. Note that the *likes* \rightarrow *bagels* arc is indeed part of the correct dependency parse, but
 6172 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,

$$h_t^{(k)} = (s_t^{(k)}, A_t^{(k)}) \quad [11.23]$$

6173 Each hypothesis is then “expanded” by considering the set of all valid actions from the
 6174 current configuration $c_t^{(k)}$, written $\mathcal{A}(c_t^{(k)})$. This yields a large set of new hypotheses. For
 6175 each action $a\mathcal{A}(c_t^{(k)})$, we score the new hypothesis $A_t^{(k)} \oplus a$. The top k hypotheses by
 6176 this scoring metric are kept, and parsing proceeds to the next step (Zhang and Clark,
 6177 2008). Note that beam search requires a scoring function for action *sequences*, rather than
 6178 individual actions. This issue will be revisited in the next section.

6179 An example of beam search is shown in Figure 11.7, with a beam size of $K = 2$. For the
 6180 first transition, the only valid action is SHIFT, so there is only one possible configuration
 6181 at $t = 2$. From this configuration, there are three possible actions. The top two are ARC-
 6182 RIGHT and ARC-LEFT, and so the resulting hypotheses from these actions are on the beam
 6183 at $t = 3$. From these configurations, there are three possible actions each, but the best
 6184 two are expansions of the bottom hypothesis at $t = 3$. Parsing continues until $t = 5$, at
 6185 which point both hypotheses reach an accepting state. The best-scoring hypothesis is then
 6186 selected as the parse.

6187 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,

$$\hat{a} = \underset{a \in \mathcal{A}(c)}{\operatorname{argmax}} \Psi(a, c, \mathbf{w}; \boldsymbol{\theta}), \quad [11.24]$$

6188 where $\mathcal{A}(c)$ is the set of admissible actions in the current configuration c , \mathbf{w} is the input,
 6189 and Ψ is a scoring function with parameters $\boldsymbol{\theta}$ (Yamada and Matsumoto, 2003).

6190 A feature-based score can be computed, $\Psi(a, c, \mathbf{w}) = \boldsymbol{\theta} \cdot \mathbf{f}(a, c, \mathbf{w})$, using features that
 6191 may consider any aspect of the current configuration and input sequence. Typical features
 6192 for transition-based dependency parsing include: the word and part-of-speech of the top
 6193 element on the stack; the word and part-of-speech of the first, second, and third elements
 6194 on the input buffer; pairs and triples of words and parts-of-speech from the top of the
 6195 stack and the front of the buffer; the distance (in tokens) between the element on the top
 6196 of the stack and the element in the front of the input buffer; the number of modifiers of
 6197 each of these elements; and higher-order dependency features as described above in the
 6198 section on graph-based dependency parsing (see, e.g., Zhang and Nivre, 2011).

6199 Parse actions can also be scored by neural networks. For example, Chen and Manning
 6200 (2014) build a feedforward network in which the input layer consists of the concatenation
 6201 of embeddings of several words and tags:

- 6202 • the top three words on the stack, and the first three words on the buffer;
- 6203 • the first and second leftmost and rightmost children (dependents) of the top two
- 6204 words on the stack;
- 6205 • the leftmost and right most grandchildren of the top two words on the stack;
- 6206 • 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,

$$\mathbf{z} = g(\Theta^{(x \rightarrow z)} \mathbf{x}(c, \mathbf{w})) \quad [11.25]$$

$$\psi(a, c, \mathbf{w}; \boldsymbol{\theta}) = \Theta_a^{(z \rightarrow y)} \mathbf{z}, \quad [11.26]$$

6207 where the vector \mathbf{z} plays the same role as the features $f(a, c, \mathbf{w})$, but is a learned representation. Chen and Manning (2014) use a cubic elementwise activation function, $g(x) = x^3$,
 6208 so that the hidden layer models products across all triples of input features. The learning
 6209 algorithm updates the embeddings as well as the parameters of the feedforward network.
 6210

6211 11.3.3 Learning to parse

6212 Transition-based dependency parsing suffers from a mismatch between the supervision,
 6213 which comes in the form of dependency trees, and the classifier's prediction space, which
 6214 is a set of parsing actions. One solution is to create new training data by converting parse
 6215 trees into action sequences; another is to derive supervision directly from the parser's
 6216 performance.

6217 11.3.3.1 Oracle-based training

6218 A transition system can be viewed as a function from action sequences (also called **deriva-**
 6219 **tions**) to parse trees. The inverse of this function is a mapping from parse trees to deriva-
 6220 tions, which is called an **oracle**. For the arc-standard and arc-eager parsing system, an
 6221 oracle can be computed in linear time in the length of the derivation (Kübler et al., 2009,
 6222 page 32). Both the arc-standard and arc-eager transition systems suffer from **spurious**
 6223 **ambiguity**: there exist dependency parses for which multiple derivations are possible,

such as $1 \leftarrow 2 \rightarrow 3$. The oracle must choose between these different derivations. For example, the algorithm described by Kübler et al. (2009) would first create the left arc ($1 \leftarrow 2$), and then create the right arc, $(1 \leftarrow 2) \rightarrow 3$; another oracle might begin by shifting twice, 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 a 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):

$$p(a | c, \mathbf{w}) = \frac{\exp \Psi(a, c, \mathbf{w}; \boldsymbol{\theta})}{\sum_{a' \in \mathcal{A}(c)} \exp \Psi(a', c, \mathbf{w}; \boldsymbol{\theta})} \quad [11.27]$$

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^N \sum_{t=1}^{|A^{(i)}|} \log p(a_t^{(i)} | c_t^{(i)}, \mathbf{w}), \quad [11.28]$$

where $|A^{(i)}|$ is the length of the action sequence $A^{(i)}$.

Recall that beam search requires a scoring function for action sequences. Such a score can be obtained by adding the log-likelihoods (or hinge losses) across all actions in the sequence (Chen and Manning, 2014).

11.3.3.2 Global objectives

The objective in Equation 11.28 is **locally-normalized**: it is the product of normalized probabilities over individual actions. A similar characterization could be made of non-probabilistic algorithms in which hinge-loss objectives are summed over individual actions. In either case, training on individual actions can be sub-optimal with respect to global performance, due to the **label bias problem** (Lafferty et al., 2001; Andor et al., 2016).

As a stylized example, suppose that a given configuration appears 100 times in the training data, with action a_1 as the oracle action in 51 cases, and a_2 as the oracle action in the other 49 cases. However, in cases where a_2 is correct, choosing a_1 results in a cascade of subsequent errors, while in cases where a_1 is correct, choosing a_2 results in only a single error. A classifier that is trained on a local objective function will learn to always choose a_1 , but choosing a_2 would minimize the overall number of errors.

This observation motivates a global objective, such as the globally-normalized conditional likelihood,

$$p(A^{(i)} | \mathbf{w}; \boldsymbol{\theta}) = \frac{\exp \sum_{t=1}^{|A^{(i)}|} \Psi(a_t^{(i)}, c_t^{(i)}, \mathbf{w})}{\sum_{A' \in \mathbb{A}(\mathbf{w})} \exp \sum_{t=1}^{|A'|} \Psi(a'_t, c'_t, \mathbf{w})}, \quad [11.29]$$

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 this sum explicitly, using dynamic programming. In transition-based parsing, this is not possible. However, the sum can be approximated using beam search,

$$\sum_{A' \in \mathbb{A}(\mathbf{w})} \exp \sum_{t=1}^{|A'|} \Psi(a'_t, c'_t, \mathbf{w}) \approx \sum_{k=1}^K \exp \sum_{t=1}^{|A^{(k)}|} \Psi(a_t^{(k)}, c_t^{(k)}, \mathbf{w}), \quad [11.30]$$

where $A^{(k)}$ is an action sequence on a beam of size K . This gives rise to the following loss function,

$$L(\boldsymbol{\theta}) = - \sum_{t=1}^{|A^{(i)}|} \Psi(a_t^{(i)}, c_t^{(i)}, \mathbf{w}) + \log \sum_{k=1}^K \exp \sum_{t=1}^{|A^{(k)}|} \Psi(a_t^{(k)}, c_t^{(k)}, \mathbf{w}). \quad [11.31]$$

6247 The derivatives of this loss involve expectations with respect to a probability distribution
6248 over action sequences on the beam.

6249 11.3.3.3 *Early update and the incremental perceptron

6250 When learning in the context of beam search, the goal is to learn a decision function so that
6251 the gold dependency parse is always reachable from at least one of the partial derivations
6252 on the beam. (The combination of a transition system (such as beam search) and a scoring
6253 function for actions is known as a **policy**.) To achieve this, we can make an **early update**
6254 as soon as the oracle action sequence “falls off” the beam, even before a complete analysis
6255 is available (Collins and Roark, 2004; Daumé III and Marcu, 2005). The loss can be based
6256 on the best-scoring hypothesis on the beam, or the sum of all hypotheses (Huang et al.,
6257 2012).

6258 For example, consider the beam search in Figure 11.7. In the correct parse, *fish* is the
6259 head of dependency arcs to both of the other two words. In the arc-standard system,
6260 this can be achieved only by using SHIFT for the first two actions. At $t = 3$, the oracle
6261 action sequence has fallen off the beam. The parser should therefore stop, and update the
6262 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
6263 oracle sequence, and $\{A_{1:3}^{(k)}\}$ is the beam.

6264 This integration of incremental search and learning was first developed in the **incremental**
6265 **perceptron** (Collins and Roark, 2004). This method updates the parameters with
6266 respect to a hinge loss, which compares the top-scoring hypothesis and the gold action

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.

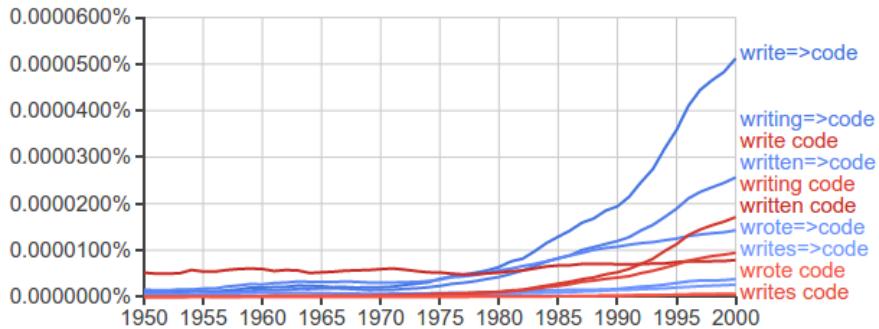


Figure 11.8: Google n-grams results for the bigram *write code* and the dependency arc *write => code* (and their morphological variants)

6267 sequence, up to the current point t . Several improvements to this basic protocol are pos-
6268 sible:

- 6269 • As noted earlier, the gold dependency parse can be derived by multiple action se-
6270 quences. Rather than checking for the presence of a single oracle action sequence on
6271 the beam, we can check if the gold dependency parse is *reachable* from the current
6272 beam, using a **dynamic oracle** (Goldberg and Nivre, 2012).
- 6273 • By maximizing the score of the gold action sequence, we are training a decision
6274 function to find the correct action given the gold context. But in reality, the parser
6275 will make errors, and the parser is not trained to find the best action given a context
6276 that may not itself be optimal. This issue is addressed by various generalizations of
6277 incremental perceptron, known as **learning to search** (Daumé III et al., 2009). Some
6278 of these methods are discussed in chapter 15.

6279 11.4 Applications

6280 Dependency parsing is used in many real-world applications: any time you want to know
6281 about pairs of words which might not be adjacent, you can use dependency arcs instead
6282 of regular expression search patterns. For example, you may want to match strings like
6283 *delicious pastries*, *delicious French pastries*, and *the pastries are delicious*.

6284 It is possible to search the Google *n*-gramscorpus by dependency edges, finding the
6285 trend in how often a dependency edge appears over time. For example, we might be inter-
6286 ested in knowing when people started talking about *writing code*, but we also want *write*
6287 *some code*, *write good code*, *write all the code*, etc. The result of a search on the dependency
6288 edge *write → code* is shown in Figure 11.8. This capability has been applied to research

6289 in digital humanities, such as the analysis of gender in Shakespeare Muralidharan and
 6290 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),

6291 which stand in some relation to each other (in this case, the relation is authorship). Such
 6292 entity pairs are often referenced via consistent chains of dependency relations. Therefore,
 6293 dependency paths are often a useful feature in supervised systems which learn to detect
 6294 new instances of a relation, based on labeled examples of other instances of the same
 6295 relation type (Culotta and Sorensen, 2004; Fundel et al., 2007; Mintz et al., 2009).

6296 Cui et al. (2005) show how dependency parsing can improve automated question an-
 6297 swering. Suppose you receive the following query:

6298 (11.1) What percentage of the nation's cheese does Wisconsin produce?

6299 The corpus contains this sentence:

6300 (11.2) In Wisconsin, where farmers produce 28% of the nation's cheese, ...

6301 The location of *Wisconsin* in the surface form of this string makes it a poor match for the
 6302 query. However, in the dependency graph, there is an edge from *produce* to *Wisconsin* in
 6303 both the question and the potential answer, raising the likelihood that this span of text is
 6304 relevant to the question.

6305 A final example comes from sentiment analysis. As discussed in chapter 4, the polarity
 6306 of a sentence can be reversed by negation, e.g.

6307 (11.3) *There is no reason at all to believe the polluters will suddenly become reasonable.*

6308 By tracking the sentiment polarity through the dependency parse, we can better iden-
 6309 tify the overall polarity of the sentence, determining when key sentiment words are re-
 6310 versed (Wilson et al., 2005; Nakagawa et al., 2010).

6311 Additional resources

6312 More details on dependency grammar and parsing algorithms can be found in the manuscript
 6313 by Kübler et al. (2009). For a comprehensive but whimsical overview of graph-based de-
 6314 pendency parsing algorithms, see Eisner (1997). Jurafsky and Martin (2018) describe an

6315 **agenda-based** version of beam search, in which the beam contains hypotheses of varying
 6316 lengths. New hypotheses are added to the beam only if their score is better than the worst
 6317 item currently on the beam. Another search algorithm for transition-based parsing is
 6318 **easy-first**, which abandons the left-to-right traversal order, and adds the highest-scoring
 6319 edges first, regardless of where they appear (Goldberg and Elhadad, 2010). Goldberg et al.
 6320 (2013) note that although transition-based methods can be implemented in linear time in
 6321 the length of the input, naïve implementations of beam search will require quadratic time,
 6322 due to the cost of copying each hypothesis when it is expanded on the beam. This issue
 6323 can be addressed by using a more efficient data structure for the stack.

6324 Exercises

- 6325 1. The dependency structure $1 \leftarrow 2 \rightarrow 3$, with 2 as the root, can be obtained from more
 6326 than one set of actions in arc-standard parsing. List both sets of actions that can
 6327 obtain this parse.
- 6328 2. This problem develops the relationship between dependency parsing and lexical-
 6329 ized context-free parsing. Suppose you have a set of unlabeled arc scores $\{\psi(i \rightarrow$
 6330 $j)\}_{i,j=1}^M \cup \{\psi(\text{ROOT} \rightarrow j)\}_{j=1}^M$.
 - 6331 a) Assuming each word type occurs no more than once in the input ($(i \neq j) \Rightarrow$
 6332 $(w_i \neq w_j)$), how would you construct a weighted lexicalized context-free gram-
 6333 mar so that the score of *any* projective dependency tree is equal to the score of
 6334 some equivalent derivation in the lexicalized context-free grammar?
 - 6335 b) Verify that your method works for the example *They fish*.
 - 6336 c) Does your method require the restriction that each word type occur no more
 6337 than once in the input? If so, why?
 - 6338 d) *If your method required that each word type occur only once in the input,
 6339 show how to generalize it.
- 6340 3. In arc-factored dependency parsing of an input of length M , the score of a parse
 6341 is the sum of M scores, one for each arc. In second order dependency parsing, the
 6342 total score is the sum over many more terms. How many terms are the score of the
 6343 parse for Figure 11.2, using a second-order dependency parser with grandparent
 6344 and sibling features? Assume that a child of ROOT has no grandparent score, and
 6345 that a node with no siblings has no sibling scores.
- 6346 4. a) In the worst case, how many terms can be involved in the score of an input of
 6347 length M , assuming second-order dependency parsing? Describe the structure
 6348 of the worst-case parse. As in the previous problem, assume that there is only
 6349 one child of ROOT, and that it does not have any grandparent scores.

- 6350 b) What about third-order dependency parsing?
- 6351 5. Provide the UD-style dependency parse for the sentence *Xi-Lan eats shoots and leaves*,
 6352 assuming *leaves* is a verb. Provide arc-standard and arc-eager derivations for this
 6353 dependency parse.
- 6354 6. Compute an upper bound on the number of successful derivations in arc-standard
 6355 shift-reduce parsing for unlabeled dependencies, as a function of the length of the
 6356 input, M . Hint: a lower bound is the number of projective decision trees, $\frac{1}{M+1} \binom{3M-2}{M-1}$ (Zhang,
 6357 2017), where $\binom{a}{b} = \frac{a!}{(a-b)!b!}$.
- 6358 7. In § 11.3.3.2, we encounter the **label bias** problem, in which a decision may be lo-
 6359 cally correct, yet lead to a cascade of errors. Design a scenario in which this occurs.
 6360 Specifically:
- 6361 • Assume an arc-standard dependency parser, whose action classifier considers
 6362 only the words at the top of the stack and at the front of the input buffer.
 - 6363 • Design two examples, which both involve a decision with identical features:
 - 6364 – In one example, shift is the correct decision; in the other example, arc-left
 6365 or arc-right is the correct decision.
 - 6366 – In one of the two examples, a mistake should lead to a cascade of attach-
 6367 ment errors.
 - 6368 – In the other example, a mistake should lead only to a single attachment
 6369 error.
- 6370 For the following exercises, run a dependency parser, such as Stanford’s CoreNLP
 6371 parser, on a large corpus of text (at least 10^5 tokens), such as `nltk.corpus.webtext`.
- 6372 8. The dependency relation NMOD:POSS indicates possession. Compute the top ten
 6373 words most frequently possessed by each of the following pronouns: *his*, *her*, *our*,
 6374 *my*, *your*, and *their* (inspired by Muralidharan and Hearst, 2013).
- 6375 9. Count all pairs of words grouped by the CONJ relation. Select all pairs of words (i, j)
 6376 for which i and j each participate in CONJ relations at least five times. Compute and
 sort by the **pointwise mutual information**, which is defined in § 14.3 as,

$$\text{PMI}(i, j) = \log \frac{\text{p}(i, j)}{\text{p}(i)\text{p}(j)}. \quad [11.32]$$

6375 Here, $\text{p}(i)$ is the fraction of CONJ relations containing word i (in either position), and
 6376 $\text{p}(i, j)$ is the fraction of such relations linking i and j (in any order).

- 6377 10. In § 4.2, we encountered lexical semantic relationships such as **synonymy** (same
6378 meaning), **antonymy** (opposite meaning), and **hyponymy** (*i* is a special case of
6379 *j*). Another relevant relation is **co-hyponymy**, which means that *i* and *j* share a
6380 hyponym. Of the top 20 pairs identified by PMI in the previous problem, how
6381 many participate in synsets that are linked by one of these four relations?

6382

Part III

6383

Meaning

6384

Chapter 12

6385

Logical semantics

6386 The previous few chapters have focused on building systems that reconstruct the **syntax**
6387 of natural language — its structural organization — through tagging and parsing. But
6388 some of the most exciting and promising potential applications of language technology
6389 involve going beyond syntax to **semantics** — the underlying meaning of the text:

- 6390 • Answering questions, such as *where is the nearest coffeeshop?* or *what is the middle name*
6391 *of the mother of the 44th President of the United States?*.
- 6392 • Building a robot that can follow natural language instructions to execute tasks.
- 6393 • Translating a sentence from one language into another, while preserving the under-
6394 lying meaning.
- 6395 • Fact-checking an article by searching the web for contradictory evidence.
- 6396 • Logic-checking an argument by identifying contradictions, ambiguity, and unsup-
6397 ported assertions.

6398 Semantic analysis involves converting natural language into a **meaning representa-**
6399 **tion**. To be useful, a meaning representation must meet several criteria:

- 6400 • **c1**: it should be unambiguous: unlike natural language, there should be exactly one
6401 meaning per statement;
- 6402 • **c2**: it should provide a way to link language to external knowledge, observations,
6403 and actions;
- 6404 • **c3**: it should support computational **inference**, so that meanings can be combined
6405 to derive additional knowledge;
- 6406 • **c4**: it should be expressive enough to cover the full range of things that people talk
6407 about in natural language.

6408 Much more than this can be said about the question of how best to represent knowledge
 6409 for computation (e.g., Sowa, 2000), but this chapter will focus on these four criteria.

6410 12.1 Meaning and denotation

6411 The first criterion for a meaning representation is that statements in the representation
 6412 should be unambiguous — they should have only one possible interpretation. Natural
 6413 language does not have this property: as we saw in chapter 10, sentences like *cats scratch*
 6414 *people with claws* have multiple interpretations.

6415 But what does it mean for a statement to be unambiguous? Programming languages
 6416 provide a useful example: the output of a program is completely specified by the rules of
 6417 the language and the properties of the environment in which the program is run. For ex-
 6418 ample, the python code $5 + 3$ will have the output 8, as will the codes $(4 * 4) - (3 * 3) + 1$
 6419 and $((8))$. This output is known as the **denotation** of the program, and can be written
 6420 as,

$$\llbracket 5+3 \rrbracket = \llbracket (4 * 4) - (3 * 3) + 1 \rrbracket = \llbracket ((8)) \rrbracket = 8. \quad [12.1]$$

6421 The denotations of these arithmetic expressions are determined by the meaning of the
 6422 **constants** (e.g., 5, 3) and the **relations** (e.g., $+$, $*$, $(,)$). Now let's consider another snippet
 6423 of python code, `double(4)`. The denotation of this code could be, $\llbracket \text{double}(4) \rrbracket = 8$, or
 6424 it could be $\llbracket \text{double}(4) \rrbracket = 44$ — it depends on the meaning of `double`. This meaning
 6425 is defined in a **world model** \mathcal{M} as an infinite set of pairs. We write the denotation with
 6426 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
 6427 model would also define the (infinite) list of constants, e.g., $\{0, 1, 2, \dots\}$. As long as the
 6428 denotation of string ϕ in model \mathcal{M} can be computed unambiguously, the language can be
 6429 said to be unambiguous.

6430 This approach to meaning is known as **model-theoretic semantics**, and it addresses
 6431 not only criterion *c1* (no ambiguity), but also *c2* (connecting language to external knowl-
 6432 edge, observations, and actions). For example, we can connect a representation of the
 6433 meaning of a statement like *the capital of Georgia* with a world model that includes knowl-
 6434 edge base of geographical facts, obtaining the denotation `Atlanta`. We might populate
 6435 a world model by applying an image analysis algorithm to Figure 12.1, and then use this
 6436 world model to evaluate **propositions** like *a man is riding a moose*. Another desirable prop-
 6437 erty of model-theoretic semantics is that when the facts change, the denotations change
 6438 too: the meaning representation of *President of the USA* would have a different denotation
 6439 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

6440 12.2 Logical representations of meaning

6441 Criterion *c3* requires that the meaning representation support inference — for example,
 6442 automatically deducing new facts from known premises. While many representations
 6443 have been proposed that meet these criteria, the most mature is the language of first-order
 6444 logic.¹

6445 12.2.1 Propositional logic

6446 The bare bones of logical meaning representation are Boolean operations on propositions:

6447 **Propositional symbols.** Greek symbols like ϕ and ψ will be used to represent **proposi-**
 6448 **tions**, which are statements that are either true or false. For example, ϕ may corre-
 6449 spond to the proposition, *bagels are delicious*.

6450 **Boolean operators.** We can build up more complex propositional formulas from Boolean
 6451 operators. These include:

- 6452 • 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).

- 6453 • Conjunction, $\phi \wedge \psi$, which is true if both ϕ and ψ are true.
- 6454 • Disjunction, $\phi \vee \psi$, which is true if at least one of ϕ and ψ is true
- 6455 • Implication, $\phi \Rightarrow \psi$, which is true unless ϕ is true and ψ is false. Implication
- 6456 has identical truth conditions to $\neg\phi \vee \psi$.
- 6457 • Equivalence, $\phi \Leftrightarrow \psi$, which is true if ϕ and ψ are both true or both false. Equiv-
- 6458 alence has identical truth conditions to $(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$.

6459 It is not strictly necessary to have all five Boolean operators: readers familiar with
 6460 Boolean logic will know that it is possible to construct all other operators from either the
 6461 NAND (not-and) or NOR (not-or) operators. Nonetheless, it is clearest to use all five
 6462 operators. From the truth conditions for these operators, it is possible to define a number
 6463 of “laws” for these Boolean operators, such as,

- 6464 • *Commutativity*: $\phi \wedge \psi = \psi \wedge \phi$, $\phi \vee \psi = \psi \vee \phi$
- 6465 • *Associativity*: $\phi \wedge (\psi \wedge \chi) = (\phi \wedge \psi) \wedge \chi$, $\phi \vee (\psi \vee \chi) = (\phi \vee \psi) \vee \chi$
- 6466 • *Complementation*: $\phi \wedge \neg\phi = \perp$, $\phi \vee \neg\phi = \top$, where \top indicates a true proposition
 6467 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,

$$\begin{aligned} \phi \Rightarrow \psi & \quad \text{If the music is loud, Max can't sleep.} \\ \phi & \quad \text{The music is loud.} \end{aligned}$$

6468 we can derive ψ (*Max can't sleep*) by application of **modus ponens**, which is one of a
 6469 set of **inference rules** that can be derived from more basic laws and used to manipulate
 6470 propositional formulas. **Automated theorem provers** are capable of applying inference
 6471 rules to a set of premises to derive desired propositions (Loveland, 2016).

6472 12.2.2 First-order logic

6473 Propositional logic is so named because it treats propositions as its base units. However,
 6474 the criterion *c4* states that our meaning representation should be sufficiently expressive.
 6475 Now consider the sentence pair,

- 6476 (12.1) If anyone is making noise, then Max can't sleep.
 6477 Abigail is making noise.

6478 People are capable of making inferences from this sentence pair, but such inferences re-
 6479 quire formal tools that are beyond propositional logic. To understand the relationship

6480 between the statement *anyone is making noise* and the statement *Abigail is making noise*, our
 6481 meaning representation requires the additional machinery of **first-order logic** (FOL).

6482 In FOL, logical propositions can be constructed from relationships between entities.
 6483 Specifically, FOL extends propositional logic with the following classes of terms:

6484 **Constants.** These are elements that name individual entities in the model, such as MAX
 6485 and ABIGAIL. The denotation of each constant in a model \mathcal{M} is an element in the
 6486 model, e.g., $[\![\text{MAX}]\!] = m$ and $[\![\text{ABIGAIL}]\!] = a$.

6487 **Relations.** Relations can be thought of as sets of entities, or sets of tuples. For example,
 6488 the relation CAN-SLEEP is defined as the set of entities who can sleep, and has the
 6489 denotation $[\![\text{CAN-SLEEP}]\!] = \{a, m, \dots\}$. To test the truth value of the proposition
 6490 CAN-SLEEP(MAX), we ask whether $[\![\text{MAX}]\!] \in [\![\text{CAN-SLEEP}]\!]$. Logical relations that are
 6491 defined over sets of entities are sometimes called **properties**.

6492 Relations may also be ordered tuples of entities. For example BROTHER(MAX,ABIGAIL)
 6493 expresses the proposition that MAX is the brother of ABIGAIL. The denotation of
 6494 such relations is a set of tuples, $[\![\text{BROTHER}]\!] = \{(m, a), (x, y), \dots\}$. To test the
 6495 truth value of the proposition BROTHER(MAX,ABIGAIL), we ask whether the tuple
 6496 $([\![\text{MAX}]\!], [\![\text{ABIGAIL}]\!])$ 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,

$$(\text{BROTHER}(\text{MAX}, \text{ABIGAIL}) \vee \text{BROTHER}(\text{MAX}, \text{STEVE})) \Rightarrow \neg \text{CAN-SLEEP}(\text{MAX}).$$

6497 This fragment of first-order logic permits only statements about specific entities. To
 6498 support inferences about statements like *If anyone is making noise, then Max can't sleep*,
 6499 two more elements must be added to the meaning representation:

6500 **Variables.** Variables are mechanisms for referring to entities that are not locally specified.
 6501 We can then write CAN-SLEEP(x) or BROTHER(x , ABIGAIL). In these cases, x is a **free**
 6502 **variable**, meaning that we have not committed to any particular assignment.

6503 **Quantifiers.** Variables are bound by quantifiers. There are two quantifiers in first-order
 6504 logic.²

- 6505 • The **existential quantifier** \exists , which indicates that there must be at least one en-
 6506 tity to which the variable can bind. For example, the statement $\exists x \text{MAKES-NOISE}(x)$
 6507 indicates that there is at least one entity for which MAKES-NOISE is true.
 6508 • The **universal quantifier** \forall , which indicates that the variable must be able to
 6509 bind to any entity in the model. For example, the statement,

$$\text{MAKES-NOISE(ABIGAIL)} \Rightarrow (\forall x \neg \text{CAN-SLEEP}(x)) \quad [12.3]$$

6510 asserts that if Abigail makes noise, no one can sleep.

6511 The expressions $\exists x$ and $\forall x$ make x into a **bound variable**. A formula that contains
 6512 no free variables is a **sentence**.

6513 **Functions.** Functions map from entities to entities, e.g., $\llbracket \text{CAPITAL-OF(GEORGIA)} \rrbracket = \llbracket \text{ATLANTA} \rrbracket$.
 6514 With functions, it is convenient to add an equality operator, supporting statements
 6515 like,

$$\forall x \exists y \text{MOTHER-OF}(x) = \text{DAUGHTER-OF}(y). \quad [12.4]$$

6516 Note that MOTHER-OF is a functional analogue of the relation MOTHER, so that
 6517 $\text{MOTHER-OF}(x) = y$ if $\text{MOTHER}(x, y)$. Any logical formula that uses functions can be
 6518 rewritten using only relations and quantification. For example,

$$\text{MAKES-NOISE}(\text{MOTHER-OF(ABIGAIL)}) \quad [12.5]$$

6519 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:

$$\forall x \exists y \text{ SPEAKS}(x, y) \quad [12.6]$$

$$\exists y \forall x \text{ SPEAKS}(x, y). \quad [12.7]$$

6520 In the first case, y may refer to several different languages, while in the second case, there
 6521 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),

$$\forall P \forall x ((\text{GOOD-BOXER}(x) \Rightarrow P(x)) \Rightarrow P(\text{BUTCH})). \quad [12.2]$$

6522 12.2.2.1 Truth-conditional semantics

6523 One way to look at the meaning of an FOL sentence ϕ is as a set of **truth conditions**,
 6524 or models under which ϕ is satisfied. But how to determine whether a sentence is true
 6525 or false in a given model? We will approach this inductively, starting with a predicate
 6526 applied to a tuple of constants. The truth of such a sentence depends on whether the
 6527 tuple of denotations of the constants is in the denotation of the predicate. For example,
 6528 CAPITAL(GEORGIA,ATLANTA) is true in model \mathcal{M} iff,

$$(\llbracket \text{GEORGIA} \rrbracket_{\mathcal{M}}, \llbracket \text{ATLANTA} \rrbracket_{\mathcal{M}}) \in \llbracket \text{CAPITAL} \rrbracket_{\mathcal{M}}. \quad [12.8]$$

6529 The Boolean operators \wedge, \vee, \dots provide ways to construct more complicated sentences,
 6530 and the truth of such statements can be assessed based on the truth tables associated with
 6531 these operators. The statement $\exists x\phi$ is true if there is some assignment of the variable x
 6532 to an entity in the model such that ϕ is true; the statement $\forall x\phi$ is true if ϕ is true under
 6533 all possible assignments of x . More formally, we would say that ϕ is **satisfied** under \mathcal{M} ,
 6534 written as $\mathcal{M} \models \phi$.

6535 Truth conditional semantics allows us to define several other properties of sentences
 6536 and pairs of sentences. Suppose that in every \mathcal{M} under which ϕ is satisfied, another
 6537 formula ψ is also satisfied; then ϕ **entails** ψ , which is also written as $\phi \models \psi$. For example,

$$\text{CAPITAL(GEORGIA,ATLANTA)} \models \exists x \text{CAPITAL(GEORGIA, } x\text{)}. \quad [12.9]$$

6538 A statement that is satisfied under any model, such as $\phi \vee \neg\phi$, is **valid**, written $\models (\phi \vee$
 6539 $\neg\phi)$. A statement that is not satisfied under any model, such as $\phi \wedge \neg\phi$, is **unsatisfiable**,
 6540 or **inconsistent**. A **model checker** is a program that determines whether a sentence ϕ
 6541 is satisfied in \mathcal{M} . A **model builder** is a program that constructs a model in which ϕ
 6542 is satisfied. The problems of checking for consistency and validity in first-order logic
 6543 are **undecidable**, meaning that there is no algorithm that can automatically determine
 6544 whether an FOL formula is valid or inconsistent.

6545 12.2.2.2 Inference in first-order logic

6546 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
 6547 can now represent such statements in first-order logic, but how are we to perform the
 6548 inference that *Max can't sleep*? One approach is to use “generalized” versions of proposi-
 6549 tional inference rules like modus ponens, which can be applied to FOL formulas. By
 6550 repeatedly applying such inference rules to a knowledge base of facts, it is possible to
 6551 produce proofs of desired propositions. To find the right sequence of inferences to derive
 6552 a desired theorem, classical artificial intelligence search algorithms like backward chain-
 6553 ing can be applied. Such algorithms are implemented in interpreters for the `prolog` logic
 6554 programming language (Pereira and Shieber, 2002).

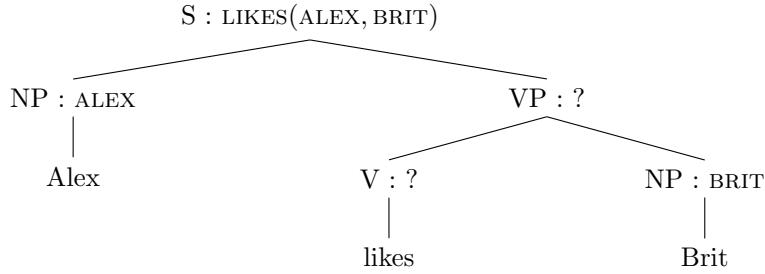


Figure 12.2: The principle of compositionality requires that we identify meanings for the constituents *likes* and *likes Brit* that will make it possible to compute the meaning for the entire sentence.

6556 12.3 Semantic parsing and the lambda calculus

6557 The previous section laid out a lot of formal machinery; the remainder of this chapter
 6558 links these formalisms back to natural language. Given an English sentence like *Alex likes*
 6559 *Brit*, how can we obtain the desired first-order logical representation, $\text{LIKES}(\text{ALEX}, \text{BRIT})$?
 6560 This is the task of **semantic parsing**. Just as a syntactic parser is a function from a natu-
 6561 ral language sentence to a syntactic structure such as a phrase structure tree, a semantic
 6562 parser is a function from natural language to logical formulas.

6563 As in syntactic analysis, semantic parsing is difficult because the space of inputs and
 6564 outputs is very large, and their interaction is complex. Our best hope is that, like syntactic
 6565 parsing, semantic parsing can somehow be decomposed into simpler sub-problems. This
 6566 idea, usually attributed to the German philosopher Gottlob Frege, is called the **principle**
 6567 **of compositionality**: the meaning of a complex expression is a function of the meanings of
 6568 that expression's constituent parts. We will define these "constituent parts" as syntactic
 6569 constituents: noun phrases and verb phrases. These constituents are combined using
 6570 function application: if the syntactic parse contains the production $x \rightarrow y z$, then the
 6571 semantics of x , written $x.\text{sem}$, will be computed as a function of the semantics of the
 6572 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).

6573 **12.3.1 The lambda calculus**

6574 Let's see how this works for a simple sentence like *Alex likes Brit*, whose syntactic structure
 6575 is shown in Figure 12.2. Our goal is the formula, LIKES(ALEX,BRIT), and it is clear that the
 6576 meaning of the constituents *Alex* and *Brit* should be ALEX and BRIT. That leaves two more
 6577 constituents: the verb *likes*, and the verb phrase *likes Brit*. The meanings of these units
 6578 must be defined in a way that makes it possible to recover the desired meaning for the
 6579 entire sentence by function application. If the meanings of *Alex* and *Brit* are constants,
 6580 then the meanings of *likes* and *likes Brit* must be functional expressions, which can be
 6581 applied to their siblings to produce the desired analyses.

6582 Modeling these partial analyses requires extending the first-order logic meaning rep-
 6583 resentation. We do this by adding **lambda expressions**, which are descriptions of anonym-
 6584 ous functions,⁵ e.g.,

$$\lambda x.\text{LIKES}(x, \text{BRIT}). \quad [12.10]$$

6585 This functional expression is the meaning of the verb phrase *likes Brit*; it takes a single
 6586 argument, and returns the result of substituting that argument for x in the expression
 6587 $\text{LIKES}(x, \text{BRIT})$. We write this substitution as,

$$(\lambda x.\text{LIKES}(x, \text{BRIT}))@\text{ALEX} = \text{LIKES}(\text{ALEX}, \text{BRIT}), \quad [12.11]$$

6588 with the symbol "@" indicating function application. Function application in the lambda
 6589 calculus is sometimes called **β -reduction** or **β -conversion**. The expression $\phi@\psi$ indicates
 6590 a function application to be performed by β -reduction, and $\phi(\psi)$ indicates a function or
 6591 predicate in the final logical form.

6592 Equation 12.11 shows how to obtain the desired semantics for the sentence *Alex likes*
 6593 *Brit*: by applying the lambda expression $\lambda x.\text{LIKES}(x, \text{BRIT})$ to the logical constant ALEX.
 6594 This rule of composition can be specified in a **syntactic-semantic grammar**, in which
 6595 syntactic productions are paired with semantic operations. For the syntactic production
 6596 $S \rightarrow NP VP$, 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.

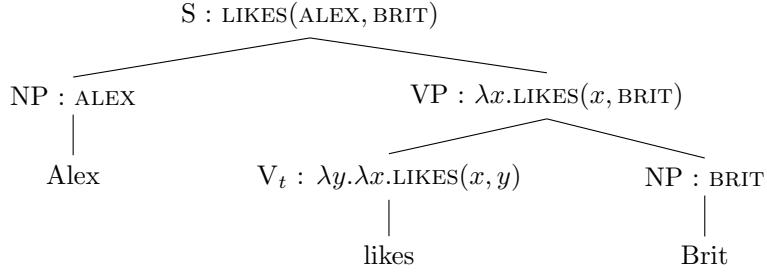


Figure 12.3: Derivation of the semantic representation for *Alex likes Brit* in the grammar G_1 .

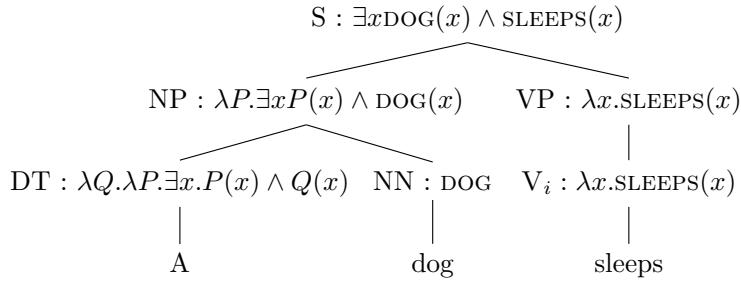
S	\rightarrow	NP VP	VP.sem@NP.sem
VP	\rightarrow	V _t NP	V _t .sem@NP.sem
VP	\rightarrow	V _i	V _i .sem
V _t	\rightarrow	likes	$\lambda y. \lambda x. \text{LIKES}(x, y)$
V _i	\rightarrow	sleeps	$\lambda x. \text{SLEEPS}(x)$
NP	\rightarrow	Alex	ALEX
NP	\rightarrow	Brit	BRIT

Table 12.1: G_1 , a minimal syntactic-semantic context-free grammar

6597 Thus, the meaning of the transitive verb *likes* is a lambda expression whose output is
 6598 *another* lambda expression: it takes y as an argument to fill in one of the slots in the LIKES
 6599 relation, and returns a lambda expression that is ready to take an argument to fill in the
 6600 other slot.⁶

6601 Table 12.1 shows a minimal syntactic-semantic grammar fragment, G_1 . The complete
 6602 **derivation** of *Alex likes Brit* in G_1 is shown in Figure 12.3. In addition to the transitive
 6603 verb *likes*, the grammar also includes the intransitive verb *sleeps*; it should be clear how
 6604 to derive the meaning of sentences like *Alex sleeps*. For verbs that can be either transitive
 6605 or intransitive, such as *eats*, we would have two terminal productions, one for each sense
 6606 (terminal productions are also called the **lexical entries**). Indeed, most of the grammar is
 6607 in the **lexicon** (the terminal productions), since these productions select the basic units of
 6608 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 6609 **12.3.2 Quantification**

6610 Things get more complicated when we move from sentences about named entities to sen-
 6611 tences that involve more general noun phrases. Let's consider the example, *A dog sleeps*,
 6612 which has the meaning $\exists x\text{DOG}(x) \wedge \text{SLEEPS}(x)$. Clearly, the DOG relation will be intro-
 6613 duced by the word *dog*, and the SLEEP relation will be introduced by the word *sleeps*.⁷
 6614 The existential quantifier \exists must be introduced by the lexical entry for the determiner *a*.⁷
 6615 However, this seems problematic for the compositional approach taken in the grammar
 6616 G_1 : if the semantics of the noun phrase *a dog* is an existentially quantified expression, how
 6617 can it be the argument to the semantics of the verb *sleeps*, which expects an entity? And
 6618 where does the logical conjunction come from?

6619 There are a few different approaches to handling these issues.⁸ We will begin by re-
 6620 versing the semantic relationship between subject NPs and VPs, so that the production
 6621 $S \rightarrow \text{NP VP}$ has the semantics $\text{NP.sem}@\text{VP.sem}$: the meaning of the sentence is now the
 6622 semantics of the noun phrase applied to the verb phrase. The implications of this change
 6623 are best illustrated by exploring the derivation of the example, shown in Figure 12.4. Let's
 6624 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,

$$\text{NP.sem} = \text{DET.sem} @ \text{NN.sem} \quad [12.16]$$

$$= (\lambda P. \lambda Q. \exists x P(x) \wedge Q(x)) @ (\text{DOG}) \quad [12.17]$$

$$= \lambda Q. \exists x \text{DOG}(x) \wedge Q(x). \quad [12.18]$$

6625 This is a lambda expression that is expecting another relation, Q , which will be provided
 6626 by the verb phrase, SLEEPS. This gives the desired analysis, $\exists x \text{DOG}(x) \wedge \text{SLEEPS}(x)$.⁹

6627 If noun phrases like *a dog* are interpreted as lambda expressions, then proper nouns
 6628 like *Alex* must be treated in the same way. This is achieved by **type-raising** from
 6629 constants to lambda expressions, $x \Rightarrow \lambda P. P(x)$. After type-raising, the semantics of *Alex* is
 6630 $\lambda P. P(\text{ALEX})$ — a lambda expression that expects a relation to tell us something about
 6631 *ALEX*.¹⁰ Again, make sure you see how the analysis in Figure 12.4 can be applied to the
 6632 sentence *Alex sleeps*.

6633 Direct objects are handled by applying the same type-raising operation to transitive
 6634 verbs: the meaning of verbs such as *likes* is raised to,

$$\lambda P. \lambda x. P(\lambda y. \text{LIKES}(x, y)) \quad [12.19]$$

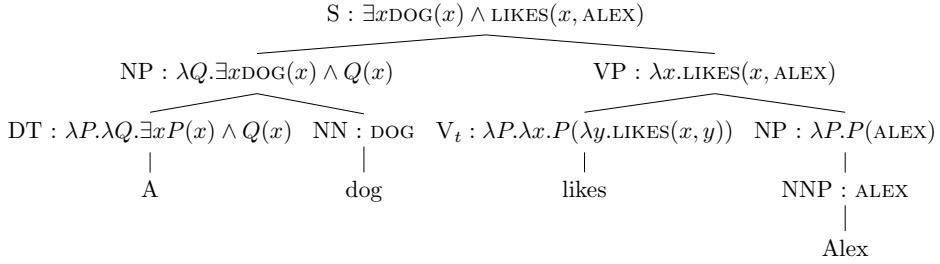
As a result, we can keep the verb phrase production $\text{VP.sem} = \text{V.sem} @ \text{NP.sem}$, knowing
 that the direct object will provide the function P in Equation 12.19. To see how this works,
 let's analyze the verb phrase *likes a dog*. After uniquely relabeling each lambda variable,
 we have,

$$\begin{aligned} \text{VP.sem} &= \text{V.sem} @ \text{NP.sem} \\ &= (\lambda P. \lambda x. P(\lambda y. \text{LIKES}(x, y))) @ (\lambda Q. \exists z \text{DOG}(z) \wedge Q(z)) \\ &= \lambda x. (\lambda Q. \exists z \text{DOG}(z) \wedge Q(z)) @ (\lambda y. \text{LIKES}(x, y)) \\ &= \lambda x. \exists z \text{DOG}(z) \wedge (\lambda y. \text{LIKES}(x, y)) @ z \\ &= \lambda x. \exists z \text{DOG}(z) \wedge \text{LIKES}(x, z). \end{aligned}$$

6635 These changes are summarized in the revised grammar G_2 , shown in Table 12.2. Figure
 6636 12.5 shows a derivation that involves a transitive verb, an indefinite noun phrase, and
 6637 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	$\rightarrow \text{NP VP}$	NP.sem@VP.sem
VP	$\rightarrow \text{V}_t \text{NP}$	$\text{V}_t.\text{sem}@\text{NP.sem}$
VP	$\rightarrow \text{V}_i$	$\text{V}_i.\text{sem}$
NP	$\rightarrow \text{DET NN}$	DET.sem@NN.sem
NP	$\rightarrow \text{NNP}$	$\lambda P. P(\text{NNP.sem})$
DET	$\rightarrow a$	$\lambda P. \lambda Q. \exists x P(x) \wedge Q(x)$
DET	$\rightarrow \text{every}$	$\lambda P. \lambda Q. \forall x (P(x) \Rightarrow Q(x))$
V_t	$\rightarrow \text{likes}$	$\lambda P. \lambda x. P(\lambda y. \text{LIKES}(x, y))$
V_i	$\rightarrow \text{sleeps}$	$\lambda x. \text{SLEEPS}(x)$
NN	$\rightarrow \text{dog}$	DOG
NNP	$\rightarrow \text{Alex}$	ALEX
NNP	$\rightarrow \text{Brit}$	BRIT

Table 12.2: G_2 , a syntactic-semantic context-free grammar fragment, which supports quantified noun phrases

6638 12.4 Learning semantic parsers

6639 As with syntactic parsing, any syntactic-semantic grammar with sufficient coverage risks
 6640 producing many possible analyses for any given sentence. Machine learning is the dom-
 6641 inant approach to selecting a single analysis. We will focus on algorithms that learn to
 6642 score logical forms by attaching weights to features of their derivations (Zettlemoyer
 6643 and Collins, 2005). Alternative approaches include transition-based parsing (Zelle and
 6644 Mooney, 1996; Misra and Artzi, 2016) and methods inspired by machine translation (Wong
 6645 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

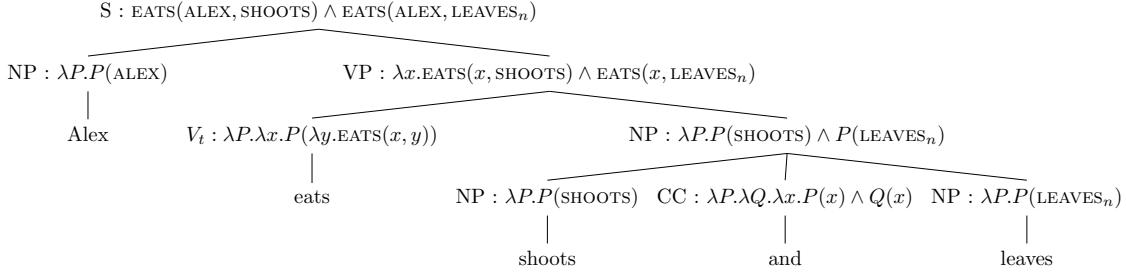
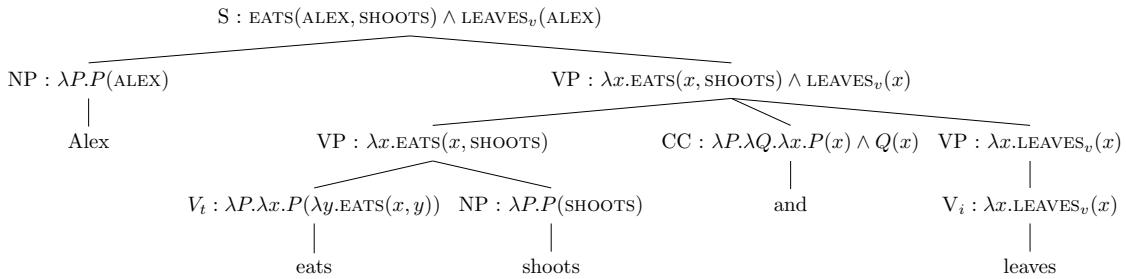
Let $w^{(i)}$ indicate a sequence of text, and let $y^{(i)}$ indicate the desired logical form. For example:

$$\begin{aligned} w^{(i)} &= \text{Alex eats shoots and leaves} \\ y^{(i)} &= \text{EATS(ALEX,SHOOTS)} \wedge \text{EATS(ALEX,LEAVES)} \end{aligned}$$

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*

6678 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
 6679 z_t indicates a single syntactic-semantic production. For example, we might have a feature
 6680 for the production $S \rightarrow NP VP : NP.sem@VP.sem$, as well as for terminal productions
 6681 like $NNP \rightarrow Alex : ALEX$. Under this decomposition, it is possible to compute scores
 6682 for each semantically-annotated subtree in the analysis of \mathbf{w} , so that bottom-up parsing
 6683 algorithms like CKY (§ 10.1) can be applied to find the best-scoring semantic analysis.

6684 Figure 12.6 shows a derivation of the correct semantic analysis of the sentence *Alex*
 6685 *eats shoots and leaves*, in a simplified grammar in which the plural noun phrases *shoots*
 6686 and *leaves* are interpreted as logical constants *SHOOTS* and *LEAVES_n*. Figure 12.7 shows a
 6687 derivation of an incorrect analysis. Assuming one feature per production, the perceptron
 6688 update is shown in Table 12.3. From this update, the parser would learn to prefer the
 6689 noun interpretation of *leaves* over the verb interpretation. It would also learn to prefer
 6690 noun phrase coordination over verb phrase coordination.

6691 While the update is explained in terms of the perceptron, it would be easy to replace
 6692 the perceptron with a conditional random field. In this case, the online updates would be
 6693 based on feature expectations, which can be computed using the inside-outside algorithm
 6694 (§ 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)

6695 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,

$$p(y, z | w) = \frac{\exp(\theta \cdot f(w, z, y))}{\sum_{y', z'} \exp(\theta \cdot f(w, z', y'))}, \quad [12.20]$$

6696 which is the standard log-linear model, applied to the logical form y and the derivation
6697 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,

$$\log \sum_z p(y, z | w) = \log \sum_z \frac{\exp(\theta \cdot f(w, z, y))}{\sum_{y', z'} \exp(\theta \cdot f(w, z', y'))} \quad [12.22]$$

$$\propto \log \sum_z \exp(\theta \cdot f(w, z', y')) \quad [12.23]$$

$$\geq \max_z \theta \cdot f(w, z, y). \quad [12.24]$$

6698 It is impossible to push the log term inside the sum over z , so our usual linear scoring
6699 function does not apply. We can recover this scoring function only in approximation, by
6700 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,

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^N \log p(\mathbf{y}^{(i)} \mid \mathbf{w}^{(i)}; \boldsymbol{\theta}) \quad [12.25]$$

$$= \sum_{i=1}^N \log \sum_z p(\mathbf{y}^{(i)}, \mathbf{z}^{(i)} \mid \mathbf{w}^{(i)}; \boldsymbol{\theta}). \quad [12.26]$$

6701 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
6702 initialization.
6703

The derivative of Equation 12.26 is,

$$\frac{\partial \ell_i}{\partial \boldsymbol{\theta}} = \sum_z p(z \mid \mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) \mathbf{f}(\mathbf{w}, z, \mathbf{y}) - \sum_{z'} p(z' \mid \mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) \mathbf{f}(\mathbf{w}, z', \mathbf{y}) \quad [12.27]$$

$$= E_{z|\mathbf{y}, \mathbf{w}} \mathbf{f}(\mathbf{w}, z, \mathbf{y}) - E_{y, z|\mathbf{w}} \mathbf{f}(\mathbf{w}, z, \mathbf{y}) \quad [12.28]$$

6704 Both expectations can be computed via bottom-up algorithms like inside-outside. Alternatively, we can again maximize rather than marginalize over derivations for an ap-
6705 proximate solution. In either case, the first term of the gradient requires us to identify
6706 derivations z that are compatible with the logical form \mathbf{y} . This can be done in a bottom-
6707 up dynamic programming algorithm, by having each cell in the table $t[i, j, X]$ include the
6708 set of all possible logical forms for $X \rightsquigarrow \mathbf{w}_{i+1:j}$. The resulting table may therefore be much
6709 larger than in syntactic parsing. This can be controlled by using pruning to eliminate in-
6710 termediate analyses that are incompatible with the final logical form \mathbf{y} (Zettlemoyer and
6711 Collins, 2005), or by using beam search and restricting the size of each cell to some fixed
6712 constant (Liang et al., 2013).
6713

6714 If we replace each expectation in Equation 12.28 with argmax and then apply stochastic
6715 gradient descent to learn the weights, we obtain the **latent variable perceptron**, a simple
6716 and general algorithm for learning with missing data. The algorithm is shown in its most
6717 basic form in Algorithm 16, but the usual tricks such as averaging and margin loss can
6718 be applied (Yu and Joachims, 2009). Aside from semantic parsing, the latent variable
6719 perceptron has been used in tasks such as machine translation (Liang et al., 2006) and
6720 named entity recognition (Sun et al., 2009). In **latent conditional random fields**, we use
6721 the full expectations rather than maximizing over the hidden variable. This model has
6722 also been employed in a range of problems beyond semantic parsing, including parse
6723 reranking (Koo and Collins, 2005) and gesture recognition (Quattoni et al., 2007).

6724 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}

6725 Similarly, in a robotic control setting, the denotation of a command would be an action or
6726 sequence of actions (Artzi and Zettlemoyer, 2013). In both cases, the idea is to reward the
6727 semantic parser for choosing an analysis whose denotation is correct: the right answer to
6728 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,

$$\ell^{(i)}(\boldsymbol{\theta}) = \log \sum_{\mathbf{y}} v_i(\mathbf{y}) \times p(\mathbf{y} \mid \mathbf{w}; \boldsymbol{\theta}) \quad [12.29]$$

$$= \log \sum_{\mathbf{y}} v_i(\mathbf{y}) \times \sum_{\mathbf{z}} p(\mathbf{y}, \mathbf{z} \mid \mathbf{w}; \boldsymbol{\theta}), \quad [12.30]$$

6729 which sums over all derivations \mathbf{z} of all valid logical forms, $\{\mathbf{y} : v_i(\mathbf{y}) = 1\}$. This cor-
6730 responds to the log-probability that the semantic parser produces a logical form with a
6731 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>

- 6760 2. Convert the following examples into first-order logic, using the relations CAN-SLEEP,
 6761 MAKES-NOISE, and BROTHER.
- 6762 • If Abigail makes noise, no one can sleep.
 6763 • If Abigail makes noise, someone cannot sleep.
 6764 • None of Abigail's brothers can sleep.
 6765 • If one of Abigail's brothers makes noise, Abigail cannot sleep.
- 6766 3. Extend the grammar fragment G_1 to include the ditransitive verb *teaches* and the
 6767 proper noun *Swahili*. Show how to derive the interpretation for the sentence *Alex*
 6768 *teaches Brit Swahili*, which should be $\text{TEACHES}(\text{ALEX}, \text{BRIT}, \text{SWAHILI})$. The grammar
 6769 need not be in Chomsky Normal Form. For the ditransitive verb, use NP_1 and NP_2
 6770 to indicate the two direct objects.
- 6771 4. Derive the semantic interpretation for the sentence *Alex likes every dog*, using gram-
 6772 mar fragment G_2 .
- 6773 5. Extend the grammar fragment G_2 to handle adjectives, so that the meaning of *an
 6774 angry dog* is $\lambda P. \exists x \text{DOG}(x) \wedge \text{ANGRY}(x) \wedge P(x)$. Specifically, you should supply the
 6775 lexical entry for the adjective *angry*, and you should specify the syntactic-semantic
 6776 productions $\text{NP} \rightarrow \text{DET } \text{NOM}$, $\text{NOM} \rightarrow \text{JJ } \text{NOM}$, and $\text{NOM} \rightarrow \text{NN}$.
- 6777 6. Extend your answer to the previous question to cover copula constructions with
 6778 predicative adjectives, such as *Alex is angry*. The interpretation should be $\text{ANGRY}(\text{ALEX})$.
 6779 You should add a verb phrase production $\text{VP} \rightarrow \text{V}_{\text{cop}} \text{ JJ}$, and a terminal production
 6780 $\text{V}_{\text{cop}} \rightarrow \text{is}$. Show why your grammar extensions result in the correct interpretation.
- 6781 7. In Figure 12.6 and Figure 12.7, we treat the plurals *shoots* and *leaves* as entities. Revise
 6782 G_2 so that the interpretation of *Alex eats leaves* is $\forall x. (\text{LEAF}(x) \Rightarrow \text{EATS}(\text{ALEX}, x))$, and
 6783 show the resulting perceptron update.
- 6784 8. Statements like *every student eats a pizza* have two possible interpretations, depend-
 6785 ing on quantifier scope:

$$\forall x \exists y \text{PIZZA}(y) \wedge (\text{STUDENT}(x) \Rightarrow \text{EATS}(x, y)) \quad [12.32]$$

$$\exists y \forall x \text{PIZZA}(y) \wedge (\text{STUDENT}(x) \Rightarrow \text{EATS}(x, y)) \quad [12.33]$$

6784 Explain why these interpretations really are different, and modify the grammar G_2
 6785 so that it can produce both interpretations.

6786 9. Derive Equation 12.27.

6787 10. In the GeoQuery domain, give a natural language query that has multiple plausible
 6788 semantic interpretations with the same denotation. List both interpretations and the
 6789 denotation.

6790 **Hint:** There are many ways to do this, but one approach involves using toponyms
6791 (place names) that could plausibly map to several different entities in the model.

6792

Chapter 13

6793

Predicate-argument semantics

6794 This chapter considers more “lightweight” semantic representations, which discard some
6795 aspects of first-order logic, but focus on predicate-argument structures. Let’s begin by
6796 thinking about the semantics of events, with a simple example:

6797 (13.1) Asha gives Boyang a book.

6798 A first-order logical representation of this sentence is,

$$\exists x. \text{BOOK}(x) \wedge \text{GIVE}(\text{ASHA}, \text{BOYANG}, x) \quad [13.1]$$

6799 In this representation, we define variable x for the book, and we link the strings *Asha* and
6800 *Boyang* to entities ASHA and BOYANG. Because the action of giving involves a giver, a
6801 recipient, and a gift, the predicate GIVE must take three arguments.

6802 Now suppose we have additional information about the event:

6803 (13.2) Yesterday, Asha reluctantly gave Boyang a book.

6804 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 ,

$$\begin{aligned} \exists e, x. & \text{GIVE-EVENT}(e) \wedge \text{GIVER}(e, \text{ASHA}) \wedge \text{GIFT}(e, x) \wedge \text{BOOK}(e, x) \wedge \text{RECIPIENT}(e, \text{BOYANG}) \\ & \wedge \text{TIME}(e, \text{YESTERDAY}) \wedge \text{MANNER}(e, \text{RELUCTANTLY}) \end{aligned}$$

6805 In this way, each argument of the event — the giver, the recipient, the gift — can be rep-
 6806 resented with a relation of its own, linking the argument to the event e . The expression
 6807 GIVER(e , ASHA) says that ASHA plays the **role** of GIVER in the event. This reformulation
 6808 handles the problem of optional information such as the time or manner of the event,
 6809 which are called **adjuncts**. Unlike arguments, adjuncts are not a mandatory part of the
 6810 relation, but under this representation, they can be expressed with additional logical rela-
 6811 tions that are conjoined to the semantic interpretation of the sentence.¹

6812 The event semantic representation can be applied to nested clauses, e.g.,

6813 (13.3) Chris sees Asha pay Boyang.

This is done by using the event variable as an argument:

$$\begin{aligned} \exists e_1 \exists e_2 \text{SEE-EVENT}(e_1) \wedge \text{SEER}(e_1, \text{CHRIS}) \wedge \text{SIGHT}(e_1, e_2) \\ \wedge \text{PAY-EVENT}(e_2) \wedge \text{PAYER}(e_2, \text{ASHA}) \wedge \text{PAYEE}(e_2, \text{BOYANG}) \end{aligned} \quad [13.3]$$

6814 As with first-order logic, the goal of event semantics is to provide a representation that
 6815 generalizes over many surface forms. Consider the following paraphrases of (13.1):

- 6816 (13.4) Asha gives a book to Boyang.
- 6817 (13.5) A book is given to Boyang by Asha.
- 6818 (13.6) A book is given by Asha to Boyang.
- 6819 (13.7) The gift of a book from Asha to Boyang ...

6820 All have the same event semantic meaning as Equation 13.1, but the ways in which the
 6821 meaning can be expressed are diverse. The final example does not even include a verb:
 6822 events are often introduced by verbs, but as shown by (13.7), the noun *gift* can introduce
 6823 the same predicate, with the same accompanying arguments.

6824 **Semantic role labeling** (SRL) is a relaxed form of semantic parsing, in which each
 6825 semantic role is filled by a set of tokens from the text itself. This is sometimes called
 6826 “shallow semantics” because, unlike model-theoretic semantic parsing, role fillers need
 6827 not be symbolic expressions with denotations in some world model. A semantic role
 6828 labeling system is required to identify all predicates, and then specify the spans of text
 6829 that fill each role. To give a sense of the task, here is a more complicated example:

- 6830 (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).

6831 In this example, there are two predicates, expressed by the verbs *want* and *give*. Thus, a
 6832 semantic role labeler might return the following output:

- 6833 • (PREDICATE : *wants*, WANTED : *Boyang*, DESIRE : *Asha to give him a linguistics book*)
 6834 • (PREDICATE : *give*, GIVER : *Asha*, RECIPIENT : *him*, GIFT : *a linguistics book*)

6835 *Boyang* and *him* may refer to the same person, but the semantic role labeling is not re-
 6836 quired to resolve this reference. Other predicate-argument representations, such as **Ab-**
 6837 **stract Meaning Representation (AMR)**, do require reference resolution. We will return to
 6838 AMR in § 13.3, but first, let us further consider the definition of semantic roles.

6839 13.1 Semantic roles

6840 In event semantics, it is necessary to specify a number of additional logical relations to
 6841 link arguments to events: GIVER, RECIPIENT, SEER, SIGHT, etc. Indeed, every predicate re-
 6842 quires a set of logical relations to express its own arguments. In contrast, adjuncts such as
 6843 TIME and MANNER are shared across many types of events. A natural question is whether
 6844 it is possible to treat mandatory arguments more like adjuncts, by identifying a set of
 6845 generic argument types that are shared across many event predicates. This can be further
 6846 motivated by examples involving related verbs:

- 6847 (13.9) Asha gave Boyang a book.
 6848 (13.10) Asha loaned Boyang a book.
 6849 (13.11) Asha taught Boyang a lesson.
 6850 (13.12) Asha gave Boyang a lesson.

6851 The respective roles of Asha, Boyang, and the book are nearly identical across the first
 6852 two examples. The third example is slightly different, but the fourth example shows that
 6853 the roles of GIVER and TEACHER can be viewed as related.

6854 One way to think about the relationship between roles such as GIVER and TEACHER is
 6855 by enumerating the set of properties that an entity typically possesses when it fulfills these
 6856 roles: givers and teachers are usually **animate** (they are alive and sentient) and **volitional**
 6857 (they choose to enter into the action).² In contrast, the thing that gets loaned or taught is
 6858 usually not animate or volitional; furthermore, it is unchanged by the event.

6859 Building on these ideas, **thematic roles** generalize across predicates by leveraging the
 6860 shared semantic properties of typical role fillers (Fillmore, 1968). For example, in exam-
 6861 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

6862 **agent.** This reflects several shared semantic properties: she is the one who is actively and
 6863 intentionally performing the action, while Boyang is a more passive participant; the book
 6864 and the lesson would play a different role, as non-animate participants in the event.

6865 Example annotations from three well known systems are shown in Figure 13.1. We
 6866 will now discuss these systems in more detail.

6867 13.1.1 VerbNet

6868 **VerbNet** (Kipper-Schuler, 2005) is a lexicon of verbs, and it includes thirty “core” thematic
 6869 roles played by arguments to these verbs. Here are some example roles, accompanied by
 6870 their definitions from the VerbNet Guidelines.³

- 6871 • AGENT: “ACTOR in an event who initiates and carries out the event intentionally or
 6872 consciously, and who exists independently of the event.”
- 6873 • PATIENT: “UNDERGOER in an event that experiences a change of state, location or
 6874 condition, that is causally involved or directly affected by other participants, and
 6875 exists independently of the event.”
- 6876 • RECIPIENT: “DESTINATION that is animate”
- 6877 • THEME: “UNDERGOER that is central to an event or state that does not have control
 6878 over the way the event occurs, is not structurally changed by the event, and/or is
 6879 characterized as being in a certain position or condition throughout the state.”
- 6880 • TOPIC: “THEME characterized by information content transferred to another partic-
 6881 ipant.”

³http://verbs.colorado.edu/verb-index/VerbNet_Guidelines.pdf

6882 VerbNet roles are organized in a hierarchy, so that a TOPIC is a type of THEME, which in
 6883 turn is a type of UNDERGOER, which is a type of PARTICIPANT, the top-level category.

6884 In addition, VerbNet organizes verb senses into a class hierarchy, in which verb senses
 6885 that have similar meanings are grouped together. Recall from § 4.2 that multiple meanings
 6886 of the same word are called **senses**, and that WordNet identifies senses for many English
 6887 words. VerbNet builds on WordNet, so that verb classes are identified by the WordNet
 6888 senses of the verbs that they contain. For example, the verb class give-13.1 includes
 6889 the first WordNet sense of *loan* and the second WordNet sense of *lend*.

6890 Each VerbNet class or subclass takes a set of thematic roles. For example, give-13.1
 6891 takes arguments with the thematic roles of AGENT, THEME, and RECIPIENT;⁴ the pred-
 6892 icate TEACH takes arguments with the thematic roles AGENT, TOPIC, RECIPIENT, and
 6893 SOURCE.⁵ So according to VerbNet, *Asha* and *Boyang* play the roles of AGENT and RECIP-
 6894 IENT in the sentences,

6895 (13.13) Asha gave Boyang a book.

6896 (13.14) Asha taught Boyang algebra.

6897 The *book* and *algebra* are both THEMES, but *algebra* is a subcategory of THEME — a TOPIC
 6898 — because it consists of information content that is given to the receiver.

6899 13.1.2 Proto-roles and PropBank

6900 Detailed thematic role inventories of the sort used in VerbNet are not universally accepted.
 6901 For example, Dowty (1991, pp. 547) notes that “Linguists have often found it hard to agree
 6902 on, and to motivate, the location of the boundary between role types.” He argues that a
 6903 solid distinction can be identified between just two **proto-roles**:

6904 **Proto-Agent.** Characterized by volitional involvement in the event or state; sentience
 6905 and/or perception; causing an event or change of state in another participant; move-
 6906 ment; exists independently of the event.

6907 **Proto-Patient.** Undergoes change of state; causally affected by another participant; sta-
 6908 tionary relative to the movement of another participant; does not exist indepen-
 6909 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.

6910 In the examples in Figure 13.1, Asha has most of the proto-agent properties: in giving
 6911 the book to Boyang, she is acting volitionally (as opposed to *Boyang got a book from Asha*, in
 6912 which it is not clear whether Asha gave up the book willingly); she is sentient; she causes a
 6913 change of state in Boyang; she exists independently of the event. Boyang has some proto-
 6914 agent properties: he is sentient and exists independently of the event. But he also has
 6915 some proto-patient properties: he is the one who is causally affected and who undergoes
 6916 change of state. The book that Asha gives Boyang has even fewer of the proto-agent
 6917 properties: it is not volitional or sentient, and it has no causal role. But it also lacks many
 6918 of the proto-patient properties: it does not undergo change of state, exists independently
 6919 of the event, and is not stationary.

6920 The **Proposition Bank**, or PropBank (Palmer et al., 2005), builds on this basic agent-
 6921 patient distinction, as a middle ground between generic thematic roles and roles that are
 6922 specific to each predicate. Each verb is linked to a list of numbered arguments, with ARG0
 6923 as the proto-agent and ARG1 as the proto-patient. Additional numbered arguments are
 6924 verb-specific. For example, for the predicate TEACH,⁷ the arguments are:

- 6925 • ARG0: the teacher
- 6926 • ARG1: the subject
- 6927 • ARG2: the student(s)

6928 Verbs may have any number of arguments: for example, WANT and GET have five, while
 6929 EAT has only ARG0 and ARG1. In addition to the semantic arguments found in the frame
 6930 files, roughly a dozen general-purpose **adjuncts** may be used in combination with any
 6931 verb. These are shown in Table 13.1.

6932 PropBank-style semantic role labeling is annotated over the entire Penn Treebank. This
 6933 annotation includes the sense of each verbal predicate, as well as the argument spans.

6934 13.1.3 FrameNet

6935 Semantic **frames** are descriptions of situations or events. Frames may be *evoked* by one
 6936 of their **lexical units** (often a verb, but not always), and they include some number of
 6937 **frame elements**, which are like roles (Fillmore, 1976). For example, the act of teaching
 6938 is a frame, and can be evoked by the verb *taught*; the associated frame elements include
 6939 the teacher, the student(s), and the subject being taught. Frame semantics has played a
 6940 significant role in the history of artificial intelligence, in the work of Minsky (1974) and
 6941 Schank and Abelson (1977). In natural language processing, the theory of frame semantics
 6942 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

6943 of roughly 1000 frames, and a corpus of more than 200,000 “exemplar sentences,” in which
 6944 the frames and their elements are annotated.⁸

6945 Rather than seeking to link semantic roles such as TEACHER and GIVER into the-
 6946 thematic roles such as AGENT, FrameNet aggressively groups verbs into frames, and links
 6947 semantically-related roles across frames. For example, the following two sentences would
 6948 be annotated identically in FrameNet:

6949 (13.15) Asha taught Boyang algebra.

6950 (13.16) Boyang learned algebra from Asha.

6951 This is because *teach* and *learn* are both lexical units in the EDUCATION-TEACHING frame.
 6952 Furthermore, roles can be shared even when the frames are distinct, as in the following
 6953 two examples:

6954 (13.17) Asha gave Boyang a book.

6955 (13.18) Boyang got a book from Asha.

6956 The GIVING and GETTING frames both have RECIPIENT and THEME elements, so Boyang
 6957 and the book would play the same role. Asha’s role is different: she is the DONOR in the
 6958 GIVING frame, and the SOURCE in the GETTING frame. FrameNet makes extensive use of
 6959 multiple inheritance to share information across frames and frame elements: for example,
 6960 the COMMERCE-SELL and LENDING frames inherit from GIVING frame.

⁸Current details and data can be found at <https://framenet.icsi.berkeley.edu/>

6961 **13.2 Semantic role labeling**

6962 The task of semantic role labeling is to identify the parts of the sentence comprising the
 6963 semantic roles. In English, this task is typically performed on the PropBank corpus, with
 6964 the goal of producing outputs in the following form:

6965 (13.19) [ARG0 Asha] [GIVE.01 gave] [ARG2 Boyang's mom] [ARG1 a book] [AM-TMP yesterday].

6966 Note that a single sentence may have multiple verbs, and therefore a given word may be
 6967 part of multiple role-fillers:

6968 (13.20) [ARG0 Asha] [WANT.01 wanted]
 Asha wanted

6969 [ARG1 Boyang to give her the book].
 [ARG0 Boyang] [GIVE.01 to give] [ARG2 her] [ARG1 the book].

6970 **13.2.1 Semantic role labeling as classification**

6971 PropBank is annotated on the Penn Treebank, and annotators used phrasal constituents
 6972 (§ 9.2.2) to fill the roles. PropBank semantic role labeling can be viewed as the task of as-
 6973 signing to each phrase a label from the set $\mathcal{R} = \{\emptyset, \text{PRED}, \text{ARG0}, \text{ARG1}, \text{ARG2}, \dots, \text{AM-LOC}, \text{AM-TMP}, \dots\}$
 6974 with respect to each predicate. If we treat semantic role labeling as a classification prob-
 6975 lem, we obtain the following functional form:

$$\hat{y}_{(i,j)} = \underset{y}{\operatorname{argmax}} \psi(\mathbf{w}, y, i, j, \rho, \tau), \quad [13.4]$$

6976 where,

- 6977 • (i, j) indicates the span of a phrasal constituent $(w_{i+1}, w_{i+2}, \dots, w_j)$;⁹
- 6978 • \mathbf{w} represents the sentence as a sequence of tokens;
- 6979 • ρ is the index of the predicate verb in \mathbf{w} ;
- 6980 • τ is the structure of the phrasal constituent parse of \mathbf{w} .

6981 Early work on semantic role labeling focused on discriminative feature-based models,
 6982 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-
 6983 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).

6984 were several systems for PropBank semantic role labeling, and their approaches and fea-
 6985 ture sets are summarized by Carreras and Márquez (2005). Typical features include: the
 6986 phrase type, head word, part-of-speech, boundaries, and neighbors of the proposed argu-
 6987 ment $w_{i+1:j}$; the word, lemma, part-of-speech, and voice of the verb w_ρ (active or passive),
 6988 as well as features relating to its frameset; the distance and path between the verb and
 6989 the proposed argument. In this way, semantic role labeling systems are high-level “con-
 6990 sumers” in the NLP stack, using features produced from lower-level components such as
 6991 part-of-speech taggers and parsers. More comprehensive feature sets are enumerated by
 6992 Das et al. (2014) and Täckström et al. (2015).

6993 A particularly powerful class of features relate to the **syntactic path** between the ar-
 6994 gument and the predicate. These features capture the sequence of moves required to get
 6995 from the argument to the verb by traversing the phrasal constituent parse of the sentence.
 6996 The idea of these features is to capture syntactic regularities in how various arguments
 6997 are realized. Syntactic path features are best illustrated by example, using the parse tree
 6998 in Figure 13.2:

- 6999 • The path from *Asha* to the verb *taught* is NNP↑NP↑S↓VP↓VBD. The first part of
 7000 the path, NNP↑NP↑S, means that we must travel up the parse tree from the NNP
 7001 tag (proper noun) to the S (sentence) constituent. The second part of the path,
 7002 S↓VP↓VBD, means that we reach the verb by producing a VP (verb phrase) from

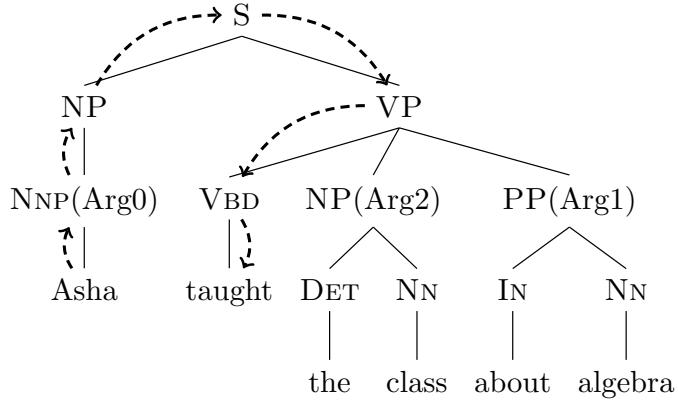


Figure 13.2: Semantic role labeling on the phrase-structure parse tree for a sentence. The dashed line indicates the syntactic path from *Asha* to the predicate verb *taught*.

7003 the S constituent, and then by producing a VBD (past tense verb). This feature is
 7004 consistent with *Asha* being in subject position, since the path includes the sentence
 7005 root S.

- 7006 • The path from *the class* to *taught* is NP↑VP↓VBD. This is consistent with *the class*
 7007 being in object position, since the path passes through the VP node that dominates
 7008 the verb *taught*.

7009 Because there are many possible path features, it can also be helpful to look at smaller
 7010 parts: for example, the upward and downward parts can be treated as separate features;
 7011 another feature might consider whether S appears anywhere in the path.

7012 Rather than using the constituent parse, it is also possible to build features from the
 7013 **dependency path** between the head word of each argument and the verb (Pradhan et al.,
 7014 2005). Using the Universal Dependency part-of-speech tagset and dependency relations (Nivre
 7015 et al., 2016), the dependency path from *Asha* to *taught* is PROPN $\xleftarrow[\text{NSUBJ}]{} \text{VERB}$, because *taught*
 7016 is the head of a relation of type $\xleftarrow[\text{NSUBJ}]{} \text{VERB}$ with *Asha*. Similarly, the dependency path from *class*
 7017 to *taught* is NOUN $\xleftarrow[\text{DOBJ}]{} \text{VERB}$, because *class* heads the noun phrase that is a direct object of
 7018 *taught*. A more interesting example is *Asha wanted to teach the class*, where the path from
 7019 *Asha* to *teach* is PROPN $\xleftarrow[\text{NSUBJ}]{} \text{VERB} \rightarrow[\text{XCOMP}] \text{VERB}$. The right-facing arrow in second relation
 7020 indicates that *wanted* is the head of its XCOMP relation with *teach*.

13.2.2 Semantic role labeling as constrained optimization

A potential problem with treating SRL as a classification problem is that there are a number of sentence-level **constraints**, which a classifier might violate.

- For a given verb, there can be only one argument of each type (ARG0, ARG1, etc.)
- Arguments cannot overlap. This problem arises when we are labeling the phrases in a constituent parse tree, as shown in Figure 13.2: if we label the PP *about algebra* as an argument or adjunct, then its children *about* and *algebra* must be labeled as \emptyset . The same constraint also applies to the syntactic ancestors of this phrase.

These constraints introduce dependencies across labeling decisions. In structure prediction problems such as sequence labeling and parsing, such dependencies are usually handled by defining a scoring over the entire structure, \mathbf{y} . Efficient inference requires that the global score decomposes into local parts: for example, in sequence labeling, the scoring function decomposes into scores of pairs of adjacent tags, permitting the application of the Viterbi algorithm for inference. But the constraints that arise in semantic role labeling are less amenable to local decomposition.¹⁰ We therefore consider **constrained 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 $\mathbf{y} \in \mathcal{C}(\tau)$,

$$\begin{aligned} \max_{\mathbf{y}} \quad & \sum_{(i,j) \in \tau} \psi(\mathbf{w}, y_{i,j}, i, j, \rho, \tau) \\ \text{s.t. } \quad & \mathbf{y} \in \mathcal{C}(\tau). \end{aligned} \quad [13.5]$$

In this formulation, the objective (shown on the first line) is a separable function of each individual labeling decision, but the constraints (shown on the second line) apply to the overall labeling. The sum $\sum_{(i,j) \in \tau}$ indicates that we are summing over all constituent spans in the parse τ . The expression s.t. in the second line means that we maximize the objective *subject to* the constraint $\mathbf{y} \in \mathcal{C}(\tau)$.

A number of practical algorithms exist for restricted forms of constrained optimization. One such restricted form is **integer linear programming**, in which the objective and constraints are linear functions of integer variables. To formulate SRL as an integer linear program, we begin by rewriting the labels as a set of binary variables $\mathbf{z} = \{z_{i,j,r}\}$ (Punyakanok 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.

7047 where $r \in \mathcal{R}$ is a label in the set $\{\text{ARG0}, \text{ARG1}, \dots, \text{AM-LOC}, \dots, \emptyset\}$. Thus, the variables
 7048 \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} .

$$\sum_{(i,j) \in \tau} \psi(\mathbf{w}, y_{i,j}, i, j, \rho, \tau) = \sum_{i,j,r} \psi(\mathbf{w}, r, i, j, \rho, \tau) \times z_{i,j,r}, \quad [13.7]$$

7049 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,

$$\forall r \neq \emptyset, \quad \sum_{(i,j) \in \tau} z_{i,j,r} \leq 1. \quad [13.8]$$

7050 Recall that $z_{i,j,r} = 1$ iff the span (i, j) has label r ; this constraint says that for each possible
 7051 label $r \neq \emptyset$, there can be at most one (i, j) such that $z_{i,j,r} = 1$. Rewriting this constraint
 7052 can be written in the form $\mathbf{A}\mathbf{z} \leq \mathbf{b}$, as you will find if you complete the exercises at the
 7053 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:

$$\forall (i, j) \in \tau, \quad \sum_{(i', j') \in \tau} \sum_{r \neq \emptyset} o((i, j), (i', j')) \times z_{i',j',r} \leq 1, \quad [13.9]$$

7054 where $o((i, j), (i, j)) = 1$.

In summary, the semantic role labeling problem can thus be rewritten as the following integer linear program,

$$\max_{\mathbf{z} \in \{0,1\}^{|\tau|}} \quad \sum_{(i,j) \in \tau} \sum_{r \in \mathcal{R}} z_{i,j,r} \psi_{i,j,r} \quad [13.10]$$

$$s.t. \quad \forall r \neq \emptyset, \quad \sum_{(i,j) \in \tau} z_{i,j,r} \leq 1. \quad [13.11]$$

$$\forall (i, j) \in \tau, \quad \sum_{(i', j') \in \tau} \sum_{r \neq \emptyset} o((i, j), (i', j')) \times z_{i',j',r} \leq 1. \quad [13.12]$$

7055 **Learning with constraints** Learning can be performed in the context of constrained op-
 7056 timization using the usual perceptron or large-margin classification updates. Because
 7057 constrained inference is generally more time-consuming, a key question is whether it is
 7058 necessary to apply the constraints during learning. Chang et al. (2008) find that better per-
 7059 formance can be obtained by learning *without* constraints, and then applying constraints
 7060 only when using the trained model to predict semantic roles for unseen data.

7061 **How important are the constraints?** Das et al. (2014) find that an unconstrained, classification-
 7062 based method performs nearly as well as constrained optimization for FrameNet parsing;
 7063 while it commits many violations of the “no-overlap” constraint, the overall F_1 score is
 7064 less than one point worse than the score at the constrained optimum. Similar results
 7065 were obtained for PropBank semantic role labeling by Punyakanok et al. (2008). He et al.
 7066 (2017) find that constrained inference makes a bigger impact if the constraints are based
 7067 on manually-labeled “gold” syntactic parses. This implies that errors from the syntac-
 7068 tic parser may limit the effectiveness of the constraints. Punyakanok et al. (2008) hedge
 7069 against parser error by including constituents from several different parsers; any con-
 7070 stituent can be selected from any parse, and additional constraints ensure that overlap-
 7071 ping constituents are not selected.

7072 **Implementation** Integer linear programming solvers such as `glpk`,¹¹ `cplex`,¹² and `Gurobi`¹³
 7073 allow inequality constraints to be expressed directly in the problem definition, rather than
 7074 in the matrix form $\mathbf{A}z \leq b$. The time complexity of integer linear programming is theoreti-
 7075 cally exponential in the number of variables $|z|$, but in practice these off-the-shelf solvers
 7076 obtain good solutions efficiently. Das et al. (2014) report that the `cplex` solver requires 43
 7077 seconds to perform inference on the FrameNet test set, which contains 4,458 predicates.

7078 Recent work has shown that many constrained optimization problems in natural lan-
 7079 guage processing can be solved in a highly parallelized fashion, using optimization tech-
 7080 niques such as **dual decomposition**, which are capable of exploiting the underlying prob-
 7081 lem structure (Rush et al., 2010). Das et al. (2014) apply this technique to FrameNet se-
 7082 mantic role labeling, obtaining an order-of-magnitude speedup over `cplex`.

7083 13.2.3 Neural semantic role labeling

7084 Neural network approaches to SRL have tended to treat it as a sequence labeling task,
 7085 using a labeling scheme such as the **BIO notation**, which we previously saw in named
 7086 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/>

7087 B-ARG1; all remaining tokens in the span are *inside*, and are therefore labeled I-ARG1.
 7088 Tokens outside any argument are labeled O. For example:

- 7089 (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,

$$\mathbf{Z}^{(1)} = \text{BiLSTM}(\mathbf{X}) \quad [13.13]$$

$$\mathbf{Z}^{(i)} = \text{BiLSTM}(\mathbf{Z}^{(i-1)}) \quad [13.14]$$

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}} \sum_{m=1}^M \Theta^{(y)} \mathbf{z}_m^{(K)} + \psi_{y_{m-1}, y_m}. \quad [13.15]$$

7090 Note that the final step maximizes over the entire labeling \mathbf{y} , and includes a score for
 7091 each tag transition ψ_{y_{m-1}, y_m} . This combination of LSTM and pairwise potentials on tags
 7092 is an example of an **LSTM-CRF**. The maximization over \mathbf{y} is performed by the Viterbi
 7093 algorithm.

7094 This model strongly outperformed alternative approaches at the time, including con-
 7095 strained decoding and convolutional neural networks.¹⁴ More recent work has combined
 7096 recurrent neural network models with constrained decoding, using the A^* search algo-
 7097 rithm to search over labelings that are feasible with respect to the constraints (He et al.,
 7098 2017). This yields small improvements over the method of Zhou and Xu (2015). He et al.
 7099 (2017) obtain larger improvements by creating an **ensemble** of SRL systems, each trained
 7100 on an 80% subsample of the corpus. The average prediction across this ensemble is more
 7101 robust than any individual model.

7102 13.3 Abstract Meaning Representation

7103 Semantic role labeling transforms the task of semantic parsing to a labeling task. Consider
 7104 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))
```

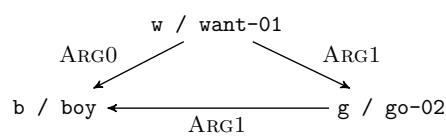


Figure 13.3: Two views of the AMR representation for the sentence *The boy wants to go.*

7105 (13.22) The boy wants to go.

7106 The PropBank semantic role labeling analysis is:

7107 • (PREDICATE : *wants*, ARG0 : *the boy*, ARG1 : *to go*)

7108 • (PREDICATE : *go*, ARG1 : *the boy*)

7109 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 7110 PENMAN notation (Matthiessen and Bateman, 1991), in which each set of parentheses 7111 introduces a variable. Each variable is an **instance** of a concept, which is indicated with 7112 the slash notation: for example, *w / want-01* indicates that the variable *w* is an instance 7113 of the concept *want-01*, which in turn refers to the PropBank frame for the first sense 7114 of the verb *want*. Relations are introduced with colons: for example, *:ARG0 (b / boy)* 7115 indicates a relation of type ARG0 with the newly-introduced variable *b*. Variables can be 7116 reused, so that when the variable *b* appears again as an argument to *g*, it is understood to 7117 refer to the same boy in both cases. This arrangement is indicated compactly in the graph 7118 structure on the right, with edges indicating concepts. 7119
7120

7121 One way in which AMR differs from PropBank-style semantic role labeling is that it 7122 reifies each entity as a variable: for example, *the boy* in (13.22) is reified in the variable 7123 *b*, which is reused as ARG0 in its relationship with *w / want-01*, and as ARG1 in its 7124 relationship with *g / go-02*. Reifying entities as variables also makes it possible to 7125 represent the substructure of noun phrases more explicitly. For example, *Asha borrowed* 7126 *the algebra book* would be represented as:

```
7127 (b / borrow-01
  :ARG0 (p / person
    :name (n / name
      :op1 "Asha"))
  :ARG1 (b2 / book
    :topic (a / algebra)))
```

7133 This indicates that the variable *p* is a person, whose name is the variable *n*; that name
 7134 has one token, the string *Asha*. Similarly, the variable *b2* is a book, and the topic of *b2*
 7135 is a variable *a* whose type is algebra. The relations name and topic are examples of
 7136 **non-core roles**, which are similar to adjunct modifiers in PropBank. However, AMR’s
 7137 inventory is more extensive, including more than 70 non-core roles, such as negation,
 7138 time, manner, frequency, and location. Lists and sequences — such as the list of tokens in
 7139 a name — are described using the roles *op1*, *op2*, etc.

7140 Another feature of AMR is that a semantic predicate can be introduced by any syntac-
 7141 tic element, as in the following examples from Banarescu et al. (2013):

- 7142 (13.23) The boy destroyed the room.
- 7143 (13.24) the destruction of the room by the boy ...
- 7144 (13.25) the boy’s destruction of the room ...

7145 All these examples have the same semantics in AMR,

```
7146 (d / destroy-01
7147   :ARG0 (b / boy)
7148   :ARG1 (r / room))
```

7149 The noun *destruction* is linked to the verb *destroy*, which is captured by the PropBank
 7150 frame *destroy-01*. This can happen with adjectives as well: in the phrase *the attractive*
 7151 *spy*, the adjective *attractive* is linked to the PropBank frame *attract-01*:

```
7152 (s / spy
7153   :ARG0-of (a / attract-01))
```

7154 In this example, *ARG0-of* is an **inverse relation**, indicating that *s* is the *ARG0* of the
 7155 predicate *a*. Inverse relations make it possible for all AMR parses to have a single root
 7156 concept, which should be the **focus** of the utterance.

7157 While AMR goes farther than semantic role labeling, it does not link semantically-
 7158 related frames such as *buy*/*sell* (as FrameNet does), does not handle quantification (as
 7159 first-order predicate calculus does), and makes no attempt to handle noun number and
 7160 verb tense (as PropBank does). A recent survey by Abend and Rappoport (2017) situ-
 7161 ates AMR with respect to several other semantic representation schemes. Other linguistic
 7162 features of AMR are summarized in the original paper (Banarescu et al., 2013) and the
 7163 tutorial slides by Schneider et al. (2015).

7164 13.3.1 AMR Parsing

7165 Abstract Meaning Representation is not a labeling of the original text — unlike PropBank
7166 semantic role labeling, and most of the other tagging and parsing tasks that we have
7167 encountered thus far. The AMR for a given sentence may include multiple concepts for
7168 single words in the sentence: as we have seen, the sentence *Asha likes algebra* contains both
7169 person and name concepts for the word *Asha*. Conversely, words in the sentence may not
7170 appear in the AMR: in *Boyang made a tour of campus*, the **light verb** *make* would not appear
7171 in the AMR, which would instead be rooted on the predicate *tour*. As a result, AMR
7172 is difficult to parse, and even evaluating AMR parsing involves considerable algorithmic
7173 complexity (Cai and Yates, 2013).

7174 A further complexity is that AMR labeled datasets do not explicitly show the **align-**
7175 **ment** between the AMR annotation and the words in the sentence. For example, the link
7176 between the word *wants* and the concept *want-01* is not annotated. To acquire train-
7177 ing data for learning-based parsers, it is therefore necessary to first perform an alignment
7178 between the training sentences and their AMR parses. Flanigan et al. (2014) introduce a
7179 rule-based parser, which links text to concepts through a series of increasingly high-recall
7180 steps.

7181 **Graph-based parsing** One family of approaches to AMR parsing is similar to the graph-
7182 based methods that we encountered in syntactic dependency parsing (chapter 11). For
7183 these systems (Flanigan et al., 2014), parsing is a two-step process:

- 7184 1. **Concept identification** (Figure 13.4a). This involves constructing concept subgraphs
7185 for individual words or spans of adjacent words. For example, in the sentence,
7186 *Asha likes algebra*, we would hope to identify the minimal subtree including just the
7187 concept *like-01* for the word *like*, and the subtree (*p / person :name (n /*
7188 *name :op1 Asha)*) for the word *Asha*.
- 7189 2. **Relation identification** (Figure 13.4b). This involves building a directed graph over
7190 the concepts, where the edges are labeled by the relation type. AMR imposes a
7191 number of constraints on the graph: all concepts must be included, the graph must
7192 be **connected** (there must be a path between every pair of nodes in the undirected
7193 version of the graph), and every node must have at most one outgoing edge of each
7194 type.

7195 Both of these problems are solved by structure prediction. Concept identification re-
7196 quires simultaneously segmenting the text into spans, and labeling each span with a graph
7197 fragment containing one or more concepts. This is done by computing a set of features
7198 for each candidate span *s* and concept labeling *c*, and then returning the labeling with the
7199 highest overall score.

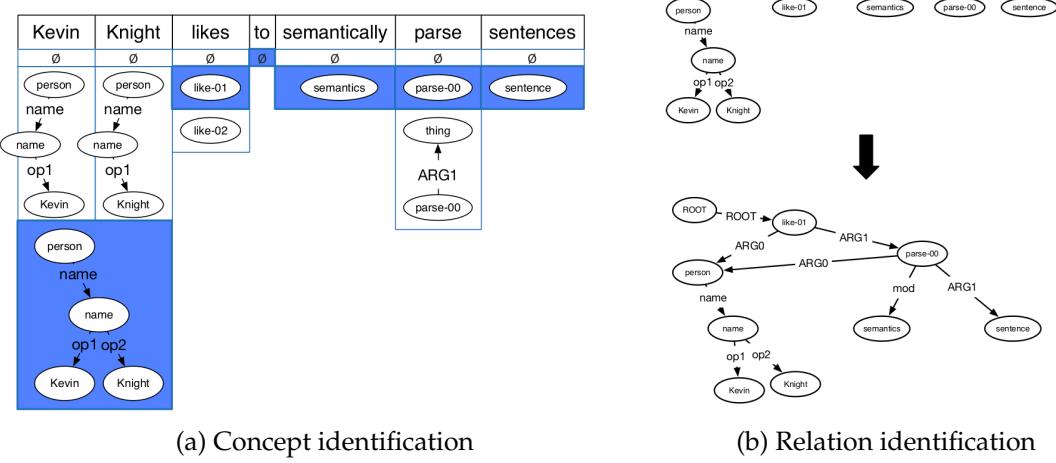


Figure 13.4: Subtasks for Abstract Meaning Representation parsing, from Schneider et al. (2015). [todo: permission]

7200 Relation identification can be formulated as search for the maximum spanning sub-
 7201 graph, under a set of constraints. Each labeled edge has a score, which is computed
 7202 from features of the concepts. We then search for the set of labeled edges that maximizes
 7203 the sum of these scores, under the constraint that the resulting graph is a well-formed
 7204 AMR (Flanigan et al., 2014). This constrained search can be performed by optimization
 7205 techniques such as integer linear programming, as described in § 13.2.2.

7206 **Transition-based parsing** In many cases, AMR parses are structurally similar to syntac-
 7207 tic dependency parses. Figure 13.5 shows one such example. This motivates an alter-
 7208 native approach to AMR parsing: modify the syntactic dependency parse until it looks
 7209 like a good AMR parse. Wang et al. (2015) propose a transition-based method, based on
 7210 incremental modifications to the syntactic dependency tree (transition-based dependency
 7211 parsing is discussed in § 11.3). At each step, the parser performs an action: for example,
 7212 adding an AMR relation label to the current dependency edge, swapping the direction of
 7213 a syntactic dependency edge, or cutting an edge and reattaching the orphaned subtree to
 7214 a new parent. The overall system is trained as a classifier, learning to choose the action as
 7215 would be given by an **oracle** that is capable of reproducing the ground-truth parse.

7216 13.4 Applications of Predicate-Argument Semantics

7217 **Question answering** Factoid questions have answers that are single words or phrases,
 7218 such as *who discovered prions?*, *where was Barack Obama born?*, and *in what year did the Knicks*
 7219 *last win the championship?* Semantic role labeling can be used to answer such questions,

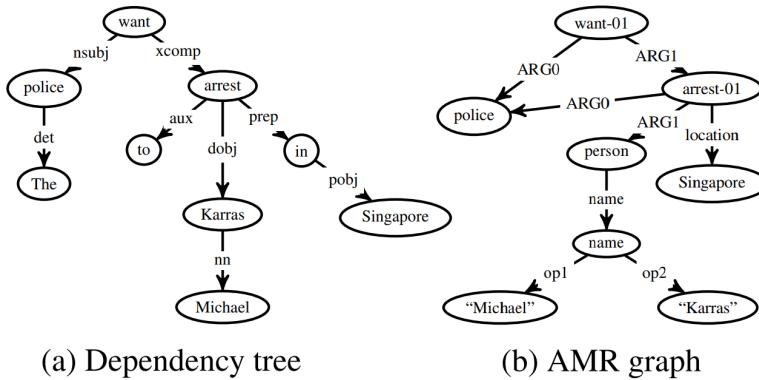


Figure 13.5: Syntactic dependency parse and AMR graph for the sentence *The police want to arrest Michael Karras in Singapore* (borrowed from Wang et al. (2015)) [todo: permission]

7220 by linking questions to sentences in a corpus of text. Shen and Lapata (2007) perform
 7221 FrameNet semantic role labeling on the query, and then construct a weighted **bipartite**
 7222 **graph**¹⁵ between FrameNet semantic roles and the words and phrases in the sentence.
 7223 This is done by first scoring all pairs of semantic roles and assignments, as shown in the
 7224 top half of Figure 13.6. They then find the bipartite edge cover, which is the minimum
 7225 weighted subset of edges such that each vertex has at least one edge, as shown in the
 7226 bottom half of Figure 13.6. After analyzing the question in this manner, Shen and Lapata
 7227 then find semantically-compatible sentences in the corpus, by performing graph matching
 7228 on the bipartite graphs for the question and candidate answer sentences. Finally, the
 7229 *expected answer phrase* in the question — typically the *wh*-word — is linked to a phrase in
 7230 the candidate answer source, and that phrase is returned as the answer.

7231 **Relation extraction** The task of **relation extraction** involves identifying pairs of entities
 7232 for which a given semantic relation holds (see § 17.2. For example, we might like to find
 7233 all pairs (i, j) such that i is the INVENTOR-OF j . PropBank semantic role labeling can
 7234 be applied to this task by identifying sentences whose verb signals the desired relation,
 7235 and then extracting ARG1 and ARG2 as arguments. (To fully solve this task, these argu-
 7236 ments must then be linked to entities, as described in chapter 17.) Christensen et al. (2010)
 7237 compare a semantic role labeling system against a simpler approach based on surface pat-
 7238 terns (Banko et al., 2007). They find that the SRL system is considerably more accurate,
 7239 but that it is several orders of magnitude slower. Conversely, Barnickel et al. (2009) apply
 7240 SENNA, a convolutional neural network SRL system (Collobert and Weston, 2008) to the
 7241 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.

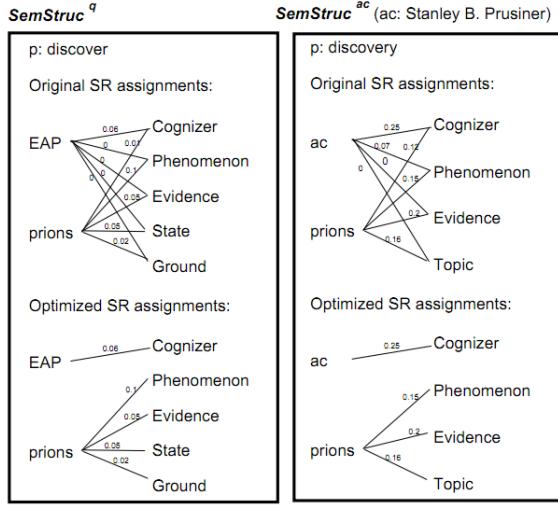


Figure 13.6: FrameNet semantic role labeling is used in factoid question answering, by aligning the semantic roles in the question (q) against those of sentences containing answer candidates (ac). “EAP” is the expected answer phrase, replacing the word *who* in the question. Figure reprinted from Shen and Lapata (2007) [todo: permission]

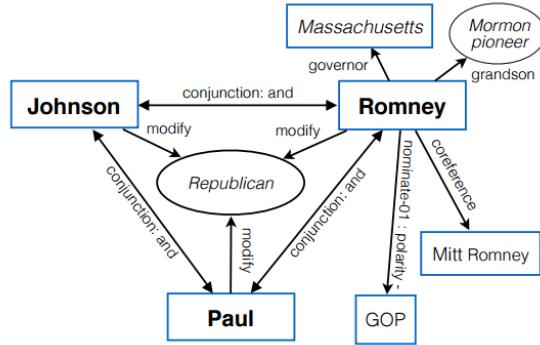


Figure 13.7: Fragment of AMR knowledge network for entity linking. Figure reprinted from Pan et al. (2015) [todo: permission]

7242 In comparison with a strong baseline that applies a set of rules to syntactic dependency
 7243 structures (Fundel et al., 2007), the SRL system is faster but less accurate. One possible
 7244 explanation for these divergent results is that Fundel et al. compare against a baseline
 7245 which is carefully tuned for performance in a relatively narrow domain, while the system
 7246 of Banko et al. is designed to analyze text across the entire web.

7247 **Entity linking** Another core task in information extraction is to link mentions of entities
7248 (e.g., *Republican candidates like Romney, Paul, and Johnson* ...) to entities in a knowledge
7249 base (e.g., LYNDON JOHNSON or GARY JOHNSON). This task, which is described in § 17.1,
7250 is often performed by examining nearby “collaborator” mentions — in this case, *Romney*
7251 and *Paul*. By jointly linking all such mentions, it is possible to arrive at a good overall
7252 solution. Pan et al. (2015) apply AMR to this problem. For each entity, they construct a
7253 knowledge network based on its semantic relations with other mentions within the same
7254 sentence. They then rerank a set of candidate entities, based on the overlap between
7255 the entity’s knowledge network and the semantic relations present in the sentence (Figure
7256 13.7).

7257 **Exercises**

- 7258 1. Write out an event semantic representation for the following sentences. You may
7259 make up your own predicates.
 - 7260 (13.26) *Abigail shares with Max.*
 - 7261 (13.27) *Abigail reluctantly shares a toy with Max.*
 - 7262 (13.28) *Abigail hates to share with Max.*
 - 7263 2. Find the PropBank framesets for *share* and *hate* at <http://verbs.colorado.edu/propbank/framesets-english-aliases/>, and rewrite your answers from the
7264 previous question, using the thematic roles ARG0, ARG1, and ARG2.
 - 7265 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
7266 sure about the parse, you can try an online parser such as <http://nlp.stanford.edu:8080/parser/>.
 - 7267 4. Compute the dependency path features for Abigail and Max in each of the example
7268 sentences (13.26) and (13.28) in Question 1, with respect to the verb *share*. Again, if
7269 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*
and *Max* is OBL according to the Universal Dependency treebank (version 2).
 - 7270 5. PropBank semantic role labeling includes **reference arguments**, such as,
7271
- 7272 (13.29) [AM-LOC The bed] on [R-AM-LOC which] I slept broke.¹⁶

¹⁶Example from 2013 NAACL tutorial slides by Shumin Wu

7277 The label R-AM-LOC indicates that word *which* is a reference to *The bed*, which ex-
 7278 presses the location of the event. Reference arguments must have referents: the tag
 7279 R-AM-LOC can appear only when AM-LOC also appears in the sentence. Show how
 7280 to express this as a linear constraint, specifically for the tag R-AM-LOC. Be sure to
 7281 correctly handle the case in which neither AM-LOC nor R-AM-LOC appear in the
 7282 sentence.

- 7283 6. Explain how to express the constraints on semantic role labeling in Equation 13.8
 7284 and Equation 13.9 in the general form $Az \geq b$.
- 7285 7. Download the FrameNet sample data (<https://framenet.icsi.berkeley.edu/fndrupal/fulltextIndex>), and train a bag-of-words classifier to predict the
 7286 frame that is evoked by each verb in each example. Your classifier should build
 7287 a bag-of-words from the sentence in which the frame-evoking lexical unit appears.
 7288 [**todo: Somehow limit to one or a few lexical units.**] [**todo: use NLTK if possible**]
- 7290 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
 7291 LSTM to predict tags. You will have to convert the downloaded instances to a BIO
 7292 sequence labeling representation first.
- 7294 9. Produce the AMR annotations for the following examples:

- 7295 (13.30) The girl likes the boy.
- 7296 (13.31) The girl was liked by the boy.
- 7297 (13.32) Abigail likes Maxwell Aristotle.
- 7298 (13.33) The spy likes the attractive boy.
- 7299 (13.34) The girl doesn't like the boy.
- 7300 (13.35) The girl likes her dog.

7301 For (13.32), recall that multi-token names are created using op1, op2, etc. You will
 7302 need to consult Banerjee et al. (2013) for (13.34), and Schneider et al. (2015) for
 7303 (13.35). You may assume that *her* refers to *the girl* in this example.

- 7304 10. Using an off-the-shelf PropBank SRL system,¹⁷ build a simplified question answer-
 7305 ing system in the style of Shen and Lapata (2007). Specifically, your system should
 7306 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.)

7321 Chapter 14

7322 Distributional and distributed 7323 semantics

7324 A recurring theme in natural language processing is the complexity of the mapping from
7325 words to meaning. In chapter 4, we saw that a single word form, like *bank*, can have mul-
7326 tiple meanings; conversely, a single meaning may be created by multiple surface forms,
7327 a lexical semantic relationship known as **synonymy**. Despite this complex mapping be-
7328 tween words and meaning, natural language processing systems usually rely on words
7329 as the basic unit of analysis. This is especially true in semantics: the logical and frame
7330 semantic methods from the previous two chapters rely on hand-crafted lexicons that map
7331 from words to semantic predicates. But how can we analyze texts that contain words
7332 that we haven't seen before? This chapter describes methods that learn representations
7333 of word meaning by analyzing unlabeled data, vastly improving the generalizability of
7334 natural language processing systems. The theory that makes it possible to acquire mean-
7335 ingful representations from unlabeled data is the **distributional hypothesis**.

7336 14.1 The distributional hypothesis

7337 Here's a word you may not know: *tezgüino* (the example is from Lin, 1998). If you do not
7338 know the meaning of *tezgüino*, then you are in the same situation as a natural language
7339 processing system when it encounters a word that did not appear in its training data.
7340 Now suppose you see that *tezgüino* is used in the following contexts:

- 7341 (14.1) A bottle of _____ is on the table.
- 7342 (14.2) Everybody likes _____.
- 7343 (14.3) Don't have _____ before you drive.
- 7344 (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

7345 What other words fit into these contexts? How about: *loud*, *motor oil*, *tortillas*, *choices*,
 7346 *wine*? Each row of Table 14.1 is a vector that summarizes the contextual properties for
 7347 each word, with a value of one for contexts in which the word can appear, and a value of
 7348 zero for contexts in which it cannot. Based on these vectors, we can conclude: *wine* is very
 7349 similar to *tezgüino*; *motor oil* and *tortillas* are fairly similar to *tezgüino*; *loud* is completely
 7350 different.

7351 These vectors, which we will call **word representations**, describe the **distributional**
 7352 properties of each word. Does vector similarity imply semantic similarity? This is the **dis-**
 7353 **distributional hypothesis**, stated by Firth (1957) as: “You shall know a word by the company
 7354 it keeps.” The distributional hypothesis has stood the test of time: distributional statistics
 7355 are a core part of language technology today, because they make it possible to leverage
 7356 large amounts of unlabeled data to learn about rare words that do not appear in labeled
 7357 training data.

7358 Distributional statistics have a striking ability to capture lexical semantic relationships
 7359 such as analogies. Figure 14.1 shows two examples, based on two-dimensional projections
 7360 of distributional **word embeddings**, discussed later in this chapter. In each case, word-
 7361 pair relationships correspond to regular linear patterns in this two dimensional space. No
 7362 labeled data about the nature of these relationships was required to identify this underly-
 7363 ing structure.

7364 **Distributional** semantics are computed from context statistics. **Distributed** seman-
 7365 tics are a related but distinct idea: that meaning can be represented by numerical vectors
 7366 rather than symbolic structures. Distributed representations are often estimated from dis-
 7367 tributional statistics, as in latent semantic analysis and WORD2VEC, described later in this
 7368 chapter. However, distributed representations can also be learned in a supervised fashion
 7369 from labeled data, as in the neural classification models encountered in chapter 3.

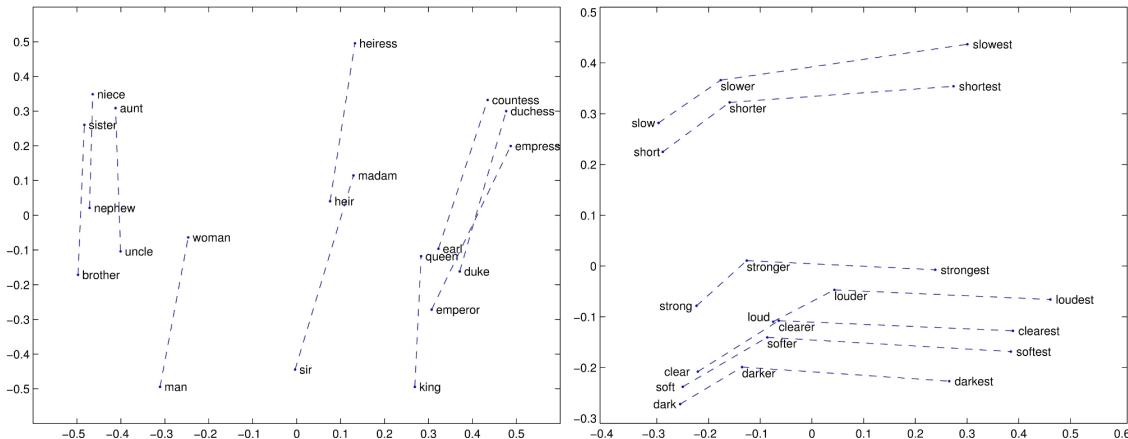


Figure 14.1: Lexical semantic relationships have regular linear structures in two dimensional projections of distributional statistics. From [http://nlp.stanford.edu/projects/glove/.\[todo: redo to make words bigger?\]](http://nlp.stanford.edu/projects/glove/.[todo: redo to make words bigger?])

14.2 Design decisions for word representations

There are many approaches for computing word representations, but most can be distinguished on three main dimensions: the nature of the representation, the source of contextual information, and the estimation procedure.

14.2.1 Representation

Today, the dominant word representations are k -dimensional vectors of real numbers, known as **word embeddings**. (The name is due to the fact that each discrete word is embedded in a continuous vector space.) This representation dates back at least to the late 1980s (Deerwester et al., 1990), and is used in popular techniques such as WORD2VEC (Mikolov et al., 2013).

Word embeddings are well suited for neural networks, where they can be plugged in as inputs. They can also be applied in linear classifiers and structure prediction models (Turian et al., 2010), although it can be difficult to learn linear models that employ real-valued features (Kummerfeld et al., 2015). A popular alternative is bit-string representations, such as **Brown clusters** (§ 14.4), in which each word is represented by a variable-length sequence of zeros and ones (Brown et al., 1992).

Another representational question is whether to estimate one embedding per surface form (e.g., *bank*), or to estimate distinct embeddings for each word sense or synset. Intuitively, if word representations are to capture the meaning of individual words, then 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*.

7390 by integrating unsupervised clustering with word embedding estimation (Huang and
 7391 Yates, 2012; Li and Jurafsky, 2015). However, Arora et al. (2016) argue that it is unnec-
 7392 essary to model distinct word senses explicitly, because the embeddings for each surface
 7393 form are a linear combination of the embeddings of the underlying senses.

7394 14.2.2 Context

7395 The distributional hypothesis says that word meaning is related to the “contexts” in which
 7396 the word appears, but context can be defined in many ways. In the *tezgiiino* example, con-
 7397 texts are entire sentences, but in practice there are far too many sentences. At the oppo-
 7398 site extreme, the context could be defined as the immediately preceding word; this is the
 7399 context considered in Brown clusters. WORD2VEC takes an intermediate approach, using
 7400 local neighborhoods of words (e.g., $h = 5$) as contexts (Mikolov et al., 2013). Contexts
 7401 can also be much larger: for example, in **latent semantic analysis**, each word’s context
 7402 vector includes an entry per document, with a value of one if the word appears in the
 7403 document (Deerwester et al., 1990); in **explicit semantic analysis**, these documents are
 7404 Wikipedia pages (Gabrilovich and Markovitch, 2007).

7405 Words in context can be labeled by their position with respect to the target word w_m
 7406 (e.g., two words before, one word after), which makes the resulting word representations
 7407 more sensitive to syntactic differences (Ling et al., 2015). Another way to incorporate
 7408 syntax is to perform parsing as a preprocessing step, and then form context vectors from
 7409 the dependency edges (Levy and Goldberg, 2014) or predicate-argument relations (Lin,
 7410 1998). The resulting context vectors for several of these methods are shown in Table 14.2.

7411 The choice of context has a profound effect on the resulting representations, which

7412 can be viewed in terms of word similarity. Applying latent semantic analysis (§ 14.3) to
 7413 contexts of size $h = 2$ and $h = 30$ yields the following nearest-neighbors for the word
 7414 *dog*:¹

- 7415 • ($h = 2$): *cat, horse, fox, pet, rabbit, pig, animal, mongrel, sheep, pigeon*
 7416 • ($h = 30$): *kennel, puppy, pet, bitch, terrier, rottweiler, canine, cat, to bark, Alsatian*

7417 Which word list is better? Each word in the $h = 2$ list is an animal, reflecting the fact that
 7418 locally, the word *dog* tends to appear in the same contexts as other animal types (e.g., *pet*
 7419 *the dog, feed the dog*). In the $h = 30$ list, nearly everything is dog-related, including specific
 7420 breeds such as *rottweiler* and *Alsatian*. The list also includes words that are not animals
 7421 (*kennel*), and in one case (*to bark*), is not a noun at all. The 2-word context window is more
 7422 sensitive to syntax, while the 30-word window is more sensitive to topic.

7423 14.2.3 Estimation

7424 Word embeddings are estimated by optimizing some objective: the likelihood of a set of
 7425 unlabeled data (or a closely related quantity), or the reconstruction of a matrix of context
 7426 counts, similar to Table 14.1.

7427 **Maximum likelihood estimation** Likelihood-based optimization is derived from the
 7428 objective $\log p(\mathbf{w}; \mathbf{U})$, where $\mathbf{U} \in \mathbb{R}^{K \times V}$ is matrix of word embeddings, and $\mathbf{w} =$
 7429 $\{\mathbf{w}_m\}_{m=1}^M$ is a corpus, represented as a list of M tokens. Recurrent neural network lan-
 7430 guage models (§ 6.3) optimize this objective directly, backpropagating to the input word
 7431 embeddings through the recurrent structure. However, state-of-the-art word embeddings
 7432 employ huge corpora with hundreds of billions of tokens, and recurrent architectures are
 7433 difficult to scale to such data. As a result, likelihood-based word embeddings are usually
 7434 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/>

7435 where $\tilde{\mathbf{C}}(\mathbf{u}, \mathbf{v})$ is the approximate reconstruction resulting from the embeddings \mathbf{u} and
 7436 \mathbf{v} , and $\|\mathbf{X}\|_F$ indicates the Frobenius norm, $\sum_{i,j} x_{i,j}^2$. Rather than factoring the matrix of
 7437 word-context counts directly, it is often helpful to transform these counts using information-
 7438 theoretic metrics such as **pointwise mutual information** (PMI), described in the next sec-
 7439 tion.

7440 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),

$$\min_{\mathbf{U} \in \mathbb{R}^{V \times K}, \mathbf{S} \in \mathbb{R}^{K \times K}, \mathbf{V} \in \mathbb{R}^{|\mathcal{C}| \times K}} \|\mathbf{C} - \mathbf{USV}^\top\|_F \quad [14.2]$$

$$\text{s.t. } \mathbf{U}^\top \mathbf{U} = \mathbb{I} \quad [14.3]$$

$$\mathbf{V}^\top \mathbf{V} = \mathbb{I} \quad [14.4]$$

$$\forall i \neq j, \mathbf{S}_{i,j} = 0, \quad [14.5]$$

7441 where V is the size of the vocabulary, $|\mathcal{C}|$ is the number of contexts, and K is size of the
 7442 resulting embeddings, which are set equal to the rows of the matrix \mathbf{U} . The matrix \mathbf{S} is
 7443 constrained to be diagonal (these diagonal elements are called the singular values), and
 7444 the columns of the product \mathbf{SV}^\top provide descriptions of the contexts. Each element $c_{i,j}$ is
 7445 then reconstructed as a **bilinear product**,

$$c_{i,j} \approx \sum_{k=1}^K u_{i,k} s_k v_{j,k}. \quad [14.6]$$

7446 The objective is to minimize the sum of squared approximation errors. The orthonormality
 7447 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
 7448 uncorrelated, so that each dimension conveys unique information. Efficient implemen-
 7449 tations of truncated singular value decomposition are available in numerical computing
 7450 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.

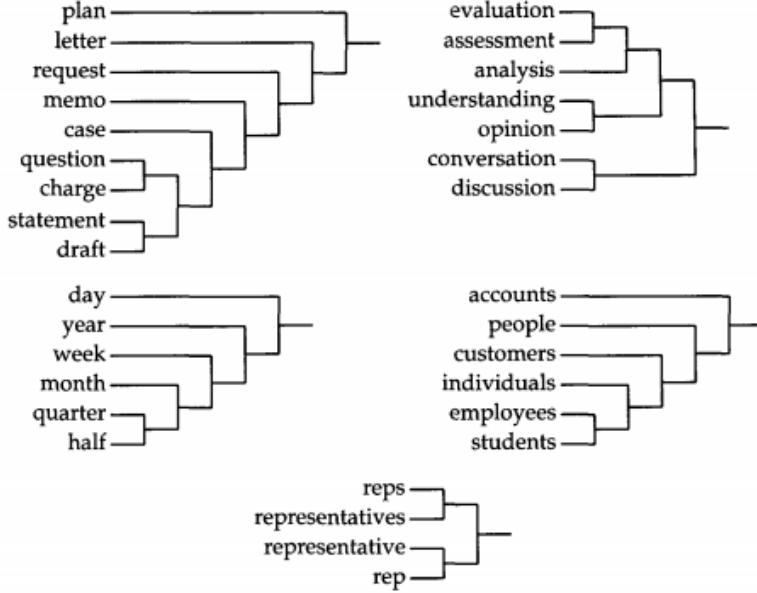


Figure 14.2: Some subtrees produced by bottom-up Brown clustering (Miller et al., 2004) on news text [todo: permission]

7475 A solution to this problem is **hierarchical clustering**: using the distributional statistics
 7476 to induce a tree-structured representation. Fragments of **Brown cluster** trees are shown in
 7477 Figure 14.2 and Table 14.3. Each word’s representation consists of a binary string describ-
 7478 ing a path through the tree: 0 for taking the left branch, and 1 for taking the right branch.
 7479 In the subtree in the upper right of the figure, the representation of the word *conversation*
 7480 is 10; the representation of the word *assessment* is 0001. Bitstring prefixes capture simila-
 7481 rity at varying levels of specificity, and it is common to use the first eight, twelve, sixteen,
 7482 and twenty bits as features in tasks such as named entity recognition (Miller et al., 2004)
 7483 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 :

$$\log p(\mathbf{w}; \mathbf{k}) \approx \sum_{m=1}^M \log p(w_m | w_{m-1}; \mathbf{k}) \quad [14.11]$$

$$\triangleq \sum_{m=1}^M \log p(w_m | k_{w_m}) + \log p(k_{w_m} | k_{w_{m-1}}). \quad [14.12]$$

7484 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.

7485 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**,

$$I(\mathbf{k}) = \sum_{k_1=1}^K \sum_{k_2=1}^K p(k_1, k_2) \times \text{PMI}(k_1, k_2) \quad [14.13]$$

$$p(k_1, k_2) = \frac{\text{count}(k_1, k_2)}{\sum_{k_{1'}=1}^K \sum_{k_{2'}=1}^K \text{count}(k_{1'}, k_{2'})},$$

7486 where $p(k_1, k_2)$ is the joint probability of a bigram involving a word in cluster k_1 followed
7487 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(., ·)},  $K$ )
    for  $i \in 1 \dots K$  do ▷ 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 ▷ 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 ▷ 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( $i, j, \mathbf{C}, \tau$ )
     $\tau \leftarrow \tau \setminus i \setminus j \cup (i, j)$  ▷ Merge the clusters in the tree
    for  $k \in \tau$  do ▷ 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  $\tau, \mathbf{C}$ 

```

7488 counts between clusters. After each merger, the co-occurrence vectors for the merged
 7489 clusters are simply added up, so that the next optimal merger can be found efficiently.

7490 This bottom-up procedure requires iterating over the entire vocabulary, and evaluating
 7491 K_t^2 possible mergers at each step, where K_t is the current number of clusters at step t
 7492 of the algorithm. Furthermore, computing the score for each merger involves a sum over
 7493 K_t^2 clusters. The maximum number of clusters is $K_0 = V$, which occurs when every word
 7494 is in its own cluster at the beginning of the algorithm. The time complexity is thus $\mathcal{O}(V^5)$.

7495 To avoid this complexity, practical implementations use a heuristic approximation
 7496 called **exchange clustering**. The K most common words are placed in clusters of their
 7497 own at the beginning of the process. We then consider the next most common word, and

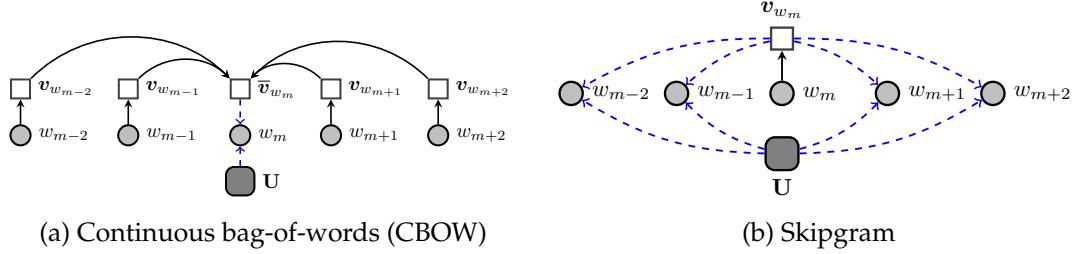


Figure 14.3: The CBOW and skipgram variants of WORD2VEC. The parameter \mathbf{U} is the matrix of word embeddings, and each \mathbf{v}_m is the context embedding for word w_m .

7498 merge it with one of the existing clusters. This continues until the entire vocabulary has
 7499 been incorporated, at which point the K clusters are merged down to a single cluster,
 7500 forming a tree. The algorithm never considers more than $K + 1$ clusters at any step, and
 7501 the complexity is $\mathcal{O}(VK + V \log V)$, with the second term representing the cost of sorting
 7502 the words at the beginning of the algorithm.

7503 14.5 Neural word embeddings

7504 Neural word embeddings combine aspects of the previous two methods: like latent se-
 7505 mantic analysis, they are a continuous vector representation; like Brown clusters, they are
 7506 trained from a likelihood-based objective. Let the vector \mathbf{u}_i represent the K -dimensional
 7507 **embedding** for word i , and let \mathbf{v}_j represent the K -dimensional embedding for context
 7508 j . The inner product $\mathbf{u}_i \cdot \mathbf{v}_j$ represents the compatibility between word i and context j .
 7509 By incorporating this inner product into an approximation to the log-likelihood of a cor-
 7510 pus, it is possible to estimate both parameters by backpropagation. WORD2VEC (Mikolov
 7511 et al., 2013) includes two such approximations: continuous bag-of-words (CBOW) and
 7512 skipgrams.

7513 14.5.1 Continuous bag-of-words (CBOW)

7514 In recurrent neural network language models, each word w_m is conditioned on a recurrently-
 7515 updated state vector, which is based on word representations going all the way back to the
 7516 beginning of the text. The **continuous bag-of-words (CBOW)** model is a simplification:
 7517 the local context is computed as an average of embeddings for words in the immediate
 7518 neighborhood $m - h, m - h + 1, \dots, m + h - 1, m + h$,

$$\bar{\mathbf{v}}_m = \frac{1}{2h} \sum_{n=1}^h \mathbf{v}_{w_{m+n}} + \mathbf{v}_{w_{m-n}}. \quad [14.14]$$

7519 Thus, CBOW is a bag-of-words model, because the order of the context words does not
 7520 matter; it is continuous, because rather than conditioning on the words themselves, we
 7521 condition on a continuous vector constructed from the word embeddings. The parameter
 7522 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,

$$\log p(\mathbf{w}) \approx \sum_{m=1}^M \log p(w_m | w_{m-h}, w_{m-h+1}, \dots, w_{m+h-1}, w_{m+h}) \quad [14.15]$$

$$= \sum_{m=1}^M \log \frac{\exp(\mathbf{u}_{w_m} \cdot \bar{\mathbf{v}}_m)}{\sum_{j=1}^V \exp(\mathbf{u}_j \cdot \bar{\mathbf{v}}_m)} \quad [14.16]$$

$$= \sum_{m=1}^M \mathbf{u}_{w_m} \cdot \bar{\mathbf{v}}_m - \log \sum_{j=1}^V \exp(\mathbf{u}_j \cdot \bar{\mathbf{v}}_m). \quad [14.17]$$

7523 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:

$$\log p(\mathbf{w}) \approx \sum_{m=1}^M \sum_{n=1}^{h_m} \log p(w_{m-n} | w_m) + \log p(w_{m+n} | w_m) \quad [14.18]$$

$$= \sum_{m=1}^M \sum_{n=1}^{h_m} \log \frac{\exp(\mathbf{u}_{w_{m-n}} \cdot \mathbf{v}_{w_m})}{\sum_{j=1}^V \exp(\mathbf{u}_j \cdot \mathbf{v}_{w_m})} + \log \frac{\exp(\mathbf{u}_{w_{m+n}} \cdot \mathbf{v}_{w_m})}{\sum_{j=1}^V \exp(\mathbf{u}_j \cdot \mathbf{v}_{w_m})} \quad [14.19]$$

$$= \sum_{m=1}^M \sum_{n=1}^{h_m} \mathbf{u}_{w_{m-n}} \cdot \mathbf{v}_{w_m} + \mathbf{u}_{w_{m+n}} \cdot \mathbf{v}_{w_m} - 2 \log \sum_{j=1}^V \exp(\mathbf{u}_j \cdot \mathbf{v}_{w_m}). \quad [14.20]$$

7524 In the skipgram approximation, each word is generated multiple times; each time it is con-
 7525 ditioned only on a single word. This makes it possible to avoid averaging the word vec-
 7526 tors, as in the CBOW model. The local neighborhood size h_m is randomly sampled from
 7527 a uniform categorical distribution over the range $\{1, 2, \dots, h_{\max}\}$; Mikolov et al. (2013) set
 7528 $h_{\max} = 10$. Because the neighborhood grows outward with h , this approach has the effect
 7529 of weighting near neighbors more than distant ones. Skipgram performs better on most
 7530 evaluations than CBOW (see § 14.6 for details of how to evaluate word representations),
 7531 but CBOW is faster to train (Mikolov et al., 2013).

7532 14.5.3 Computational complexity

7533 The WORD2VEC models can be viewed as an efficient alternative to recurrent neural net-
 7534 work language models, which involve a recurrent state update whose time complexity

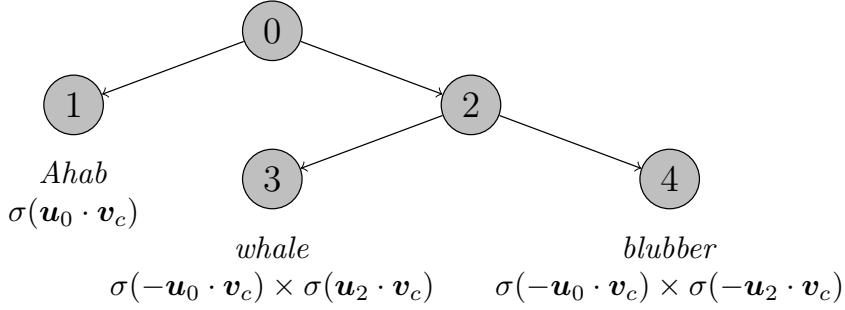


Figure 14.4: A fragment of a hierarchical softmax tree. The probability of each word is computed as a product of probabilities of local branching decisions in the tree.

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

7550 logarithmic in the number of leaf nodes, and thus the number of multiplications is equal
 7551 to $\mathcal{O}(\log V)$. The number of non-leaf nodes is equal to $\mathcal{O}(2V - 1)$, so the number of pa-
 7552 rameters to be estimated increases by only a small multiple. The tree can be constructed
 7553 using an incremental clustering procedure similar to hierarchical Brown clusters (Mnih
 7554 and Hinton, 2008), or by using the Huffman (1952) encoding algorithm for lossless com-
 7555 pression.

7556 **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**:

$$\psi(i, j) = \log \sigma(\mathbf{u}_i \cdot \mathbf{v}_j) + \sum_{i' \in \mathcal{W}_{\text{neg}}} \log(1 - \sigma(\mathbf{u}_{i'} \cdot \mathbf{v}_j)), \quad [14.23]$$

7557 where $\psi(i, j)$ is the score for word i in context j , and \mathcal{W}_{neg} is the set of negative samples.
 7558 The objective is to maximize the sum over the corpus, $\sum_{m=1}^M \psi(w_m, c_m)$, where w_m is
 7559 token m and c_m is the associated context.

7560 The set of negative samples \mathcal{W}_{neg} is obtained by sampling from a unigram language
 7561 model. Mikolov et al. (2013) construct this unigram language model by exponentiating
 7562 the empirical word probabilities, setting $\hat{p}(i) \propto (\text{count}(i))^{\frac{3}{4}}$. This has the effect of redi-
 7563 tributing probability mass from common to rare words. The number of negative samples
 7564 increases the time complexity of training by a constant factor. Mikolov et al. (2013) report
 7565 that 5-20 negative samples works for small training sets, and that two to five samples
 7566 suffice for larger corpora.

7567 **14.5.4 Word embeddings as matrix factorization**

7568 The negative sampling objective in Equation 14.23 can be justified as an efficient approx-
 7569 imation to the log-likelihood, but it is also closely linked to the matrix factorization ob-
 7570 jective employed in latent semantic analysis. For a matrix of word-context pairs in which
 7571 all counts are non-zero, negative sampling is equivalent to factorization of the matrix M ,
 7572 where $M_{ij} = \text{PMI}(i, j) - \log k$: each cell in the matrix is equal to the pointwise mutual
 7573 information of the word and context, shifted by $\log k$, with k equal to the number of neg-
 7574 ative samples (Levy and Goldberg, 2014). For word-context pairs that are not observed in
 7575 the data, the pointwise mutual information is $-\infty$, but this can be addressed by consid-
 7576 ering only PMI values that are greater than $\log k$, resulting in a matrix of **shifted positive**

7577 **pointwise mutual information,**

$$M_{ij} = \max(0, \text{PMI}(i, j) - \log k). \quad [14.24]$$

7578 Word embeddings are obtained by factoring this matrix with truncated singular value
7579 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,

$$\begin{aligned} \min_{\mathbf{u}, \mathbf{v}, b, \tilde{b}} \quad & \sum_{j=1}^V \sum_{j \in C} f(M_{ij}) \left(\widehat{\log M_{ij}} - \log M_{ij} \right)^2 \\ \text{s.t.} \quad & \widehat{\log M_{ij}} = \mathbf{u}_i \cdot \mathbf{v}_j + b_i + \tilde{b}_j, \end{aligned} \quad [14.25]$$

7580 where b_i and \tilde{b}_j are offsets for word i and context j , which are estimated jointly with the
7581 embeddings \mathbf{u} and \mathbf{v} . The weighting function $f(M_{ij})$ is set to be zero at $M_{ij} = 0$, thus
7582 avoiding the problem of taking the logarithm of zero counts; it saturates at $M_{ij} = m_{\max}$,
7583 thus avoiding the problem of overcounting common word-context pairs. This heuristic
7584 turns out to be critical to the method’s performance.

7585 The time complexity of sparse matrix reconstruction is determined by the number of
7586 non-zero word-context counts. Pennington et al. (2014) show that this number grows
7587 sublinearly with the size of the dataset: roughly $\mathcal{O}(N^{0.8})$ for typical English corpora. In
7588 contrast, the time complexity of WORD2VEC is linear in the corpus size. Computing the co-
7589 occurrence counts also requires linear time in the size of the corpus, but this operation can
7590 easily be parallelized using MapReduce-style algorithms (Dean and Ghemawat, 2008).

7591 14.6 Evaluating word embeddings

7592 Distributed word representations can be evaluated in two main ways. **Intrinsic** evalua-
7593 tions test whether the representations cohere with our intuitions about word meaning.
7594 **Extrinsic** evaluations test whether they are useful for downstream tasks, such as sequence
7595 labeling.

7596 14.6.1 Intrinsic evaluations

7597 A basic question for word embeddings is whether the similarity of words i and j is re-
7598 flected in the similarity of the vectors \mathbf{u}_i and \mathbf{u}_j . **Cosine similarity** is typically used to
7599 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)).

7600 For any embedding method, we can evaluate whether the cosine similarity of word em-
 7601 beddings is correlated with human judgments of word similarity. The WS-353 dataset (Finkel-
 7602 stein et al., 2002) includes similarity scores for 353 word pairs (Table 14.4). To test the
 7603 accuracy of embeddings for rare and morphologically complex words, Luong et al. (2013)
 7604 introduce a dataset of “rare words.” Outside of English, word similarity resources are
 7605 limited, mainly consisting of translations of WS-353.

7606 Word analogies (e.g., *king:queen :: man:woman*) have also been used to evaluate word
 7607 embeddings (Mikolov et al., 2013). In this evaluation, the system is provided with the first
 7608 three parts of the analogy ($i_1 : j_1 :: i_2 : ?$), and the final element is predicted by finding the
 7609 word embedding most similar to $\mathbf{u}_{i_1} - \mathbf{u}_{j_1} + \mathbf{u}_{i_2}$. Another evaluation tests whether word
 7610 embeddings are related to broad lexical semantic categories called **supersenses** (Caramita
 7611 and Johnson, 2003): verbs of motion, nouns that describe animals, nouns that describe
 7612 body parts, and so on. These supersenses are annotated for English synsets in Word-
 7613 Net (Fellbaum, 2010). This evaluation is implemented in the `qvec` metric, which tests
 7614 whether the matrix of supersenses can be reconstructed from the matrix of word embed-
 7615 dings (Tsvetkov et al., 2015).

7616 Levy et al. (2015) compared several dense word representations for English — includ-
 7617 ing latent semantic analysis, WORD2VEC, and GloVe — using six word similarity metrics
 7618 and two analogy tasks. None of the embeddings outperformed the others on every task,
 7619 but skipgrams were the most broadly competitive. Hyperparameter tuning played a key
 7620 role: any method will perform badly if the wrong hyperparameters are used. Relevant
 7621 hyperparameters include the embedding size, as well as algorithm-specific details such
 7622 as the neighborhood size and the number of negative samples.

7623 14.6.2 Extrinsic evaluations

7624 Word representations contribute to downstream tasks like sequence labeling and docu-
 7625 ment classification by enabling generalization across words. The use of distributed repre-
 7626 sentations as features is a form of **semi-supervised learning**, in which performance on a

7627 supervised learning problem is augmented by learning distributed representations from
 7628 unlabeled data (Miller et al., 2004; Koo et al., 2008; Turian et al., 2010). These **pre-trained**
 7629 **word representations** can be used as features in a linear prediction model, or as the input
 7630 layer in a neural network, such as a Bi-LSTM tagging model (§ 7.6). Word representations
 7631 can be evaluated by the performance of the downstream systems that consume them:
 7632 for example, GloVe embeddings are convincingly better than Latent Semantic Analysis
 7633 as features in the downstream task of named entity recognition (Pennington et al., 2014).
 7634 Unfortunately, extrinsic and intrinsic evaluations do not always point in the same direc-
 7635 tion, and the best word representations for one downstream task may perform poorly on
 7636 another task (Schnabel et al., 2015).

7637 When word representations are updated from labeled data in the downstream task,
 7638 they are said to be **fine-tuned**. When labeled data is plentiful, pre-training may be un-
 7639 necessary; when labeled data is scarce, fine-tuning may lead to overfitting. Various com-
 7640 binations of pre-training and fine-tuning can be employed. Pre-trained embeddings can
 7641 be used as initialization before fine-tuning, and this can substantially improve perfor-
 7642 mance (Lample et al., 2016). Alternatively, both fine-tuned and pre-trained embeddings
 7643 can be used as inputs in a single model (Kim, 2014).

7644 In semi-supervised scenarios, pretrained word embeddings can be replaced by “con-
 7645 textualized” word representations (Peters et al., 2018). These contextualized represen-
 7646 tations are set to the hidden states of a deep bi-directional LSTM, which is trained as a
 7647 bi-directional language model, motivating the name **ELMo (embeddings from language**
 7648 **models**). Given a supervised learning problem, the language model generates contextu-
 7649 alized representations, which are then used as the base layer in a task-specific supervised
 7650 neural network. This approach yields significant gains over pretrained word embeddings
 7651 on several tasks, presumably because the contextualized embeddings use unlabeled data
 7652 to learn how to integrate linguistic context into the base layer of the supervised neural
 7653 network.

7654 14.7 Distributed representations beyond distributional statistics

7655 Distributional word representations can be estimated from huge unlabeled datasets, thereby
 7656 covering many words that do not appear in labeled data: for example, GloVe embeddings
 7657 are estimated from 800 billion tokens of web data,³ while the largest labeled datasets for
 7658 NLP tasks are on the order of millions of tokens. Nonetheless, even a dataset of hundreds
 7659 of billions of tokens will not cover every word that may be encountered in the future.
 7660 Furthermore, many words will appear only a few times, making their embeddings un-
 7661 reliable. Many languages exceed English in morphological complexity, and thus have
 7662 lower token-to-type ratios. When this problem is coupled with small training corpora, it

³<http://commoncrawl.org/>

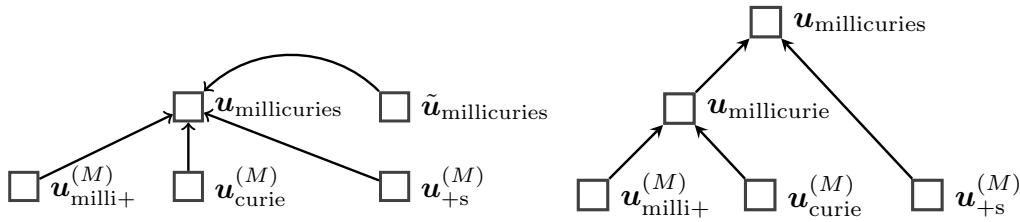


Figure 14.5: Two architectures for building word embeddings from subword units. On the left, morpheme embeddings $u^{(m)}$ are combined by addition with the non-compositional word embedding \tilde{u} (Botha and Blunsom, 2014). On the right, morpheme embeddings are combined in a recursive neural network (Luong et al., 2013).

7663 becomes especially important to leverage other sources of information beyond distributional statistics.

7665 14.7.1 Word-internal structure

7666 One solution is to incorporate word-internal structure into word embeddings. Purely
7667 distributional approaches consider words as atomic units, but in fact, many words have
7668 internal structure, so that their meaning can be **composed** from the representations of
7669 sub-word units. Consider the following terms, all of which are missing from Google's
7670 pre-trained WORD2VEC embeddings:⁴

7671 **millicuries** This word has **morphological** structure (see § 9.1.2 for more on morphology):
7672 the prefix *milli-* indicates an amount, and the suffix *-s* indicates a plural. (A *millicurie*
7673 is an unit of radioactivity.)

7674 **caesium** This word is a single morpheme, but the characters *-ium* are often associated
7675 with chemical elements. (*Caesium* is the British spelling of a chemical element,
7676 spelled *cesium* in American English.)

7677 **IAEA** This term is an acronym, as suggested by the use of capitalization. The prefix *I-* frequently
7678 refers to international organizations, and the suffix *-A* often refers to agencies or associations. (*IAEA* is the International Atomic Energy Agency.)

7680 **Zhezhgan** This term is in title case, suggesting the name of a person or place, and the
7681 character bigram *zh* indicates that it is likely a transliteration. (*Zhezhgan* is a mining
7682 facility in Kazakhstan.)

⁴<https://code.google.com/archive/p/word2vec/>, accessed September 20, 2017

7683 How can word-internal structure be incorporated into word representations? One
7684 approach is to construct word representations from embeddings of the characters or mor-
7685 phemes. For example, if word i has morphological segments \mathcal{M}_i , then its embedding can
7686 be constructed by addition (Botha and Blunsom, 2014),

$$\mathbf{u}_i = \tilde{\mathbf{u}}_i + \sum_{j \in \mathcal{M}_i} \mathbf{u}_j^{(M)}, \quad [14.27]$$

7687 where $\mathbf{u}_m^{(M)}$ is a morpheme embedding and $\tilde{\mathbf{u}}_i$ is a non-compositional embedding of the
7688 whole word, which is an additional free parameter of the model (Figure 14.5, left side).
7689 All embeddings are estimated from a **log-bilinear language model** (Mnih and Hinton,
7690 2007), which is similar to the CBOW model (§ 14.5), but includes only contextual informa-
7691 tion from preceding words. The morphological segments are obtained using an unsuper-
7692 vised segmenter (Creutz and Lagus, 2007). For words that do not appear in the training
7693 data, the embedding can be constructed directly from the morphemes, assuming that each
7694 morpheme appears in some other word in the training data. The free parameter $\tilde{\mathbf{u}}$ adds
7695 flexibility: words with similar morphemes are encouraged to have similar embeddings,
7696 but this parameter makes it possible for them to be different.

7697 Word-internal structure can be incorporated into word representations in various other
7698 ways. Here are some of the main parameters.

7699 **Subword units.** Examples like *IAEA* and *Zhezhgan* are not based on morphological com-
7700 position, and a morphological segmenter is unlikely to identify meaningful sub-
7701 word units for these terms. Rather than using morphemes for subword embeddings,
7702 one can use characters (Santos and Zadrozny, 2014; Ling et al., 2015; Kim et al., 2016),
7703 character n -grams (Wieting et al., 2016; Bojanowski et al., 2017), and **byte-pair en-**
7704 **codings**, a compression technique which captures frequent substrings (Gage, 1994;
7705 Sennrich et al., 2016).

7706 **Composition.** Combining the subword embeddings by addition does not differentiate
7707 between orderings, nor does it identify any particular morpheme as the **root**. A
7708 range of more flexible compositional models have been considered, including re-
7709 currence (Ling et al., 2015), convolution (Santos and Zadrozny, 2014; Kim et al.,
7710 2016), and **recursive neural networks** (Luong et al., 2013), in which representa-
7711 tions of progressively larger units are constructed over a morphological parse, e.g.
7712 $((milli+curie)+s)$, $((in+flam)+able)$, $(in+(vis+ible))$. A recursive embedding model is
7713 shown in the right panel of Figure 14.5.

7714 **Estimation.** Estimating subword embeddings from a full dataset is computationally ex-
7715 pensive. An alternative approach is to train a subword model to match pre-trained
7716 word embeddings (Cotterell et al., 2016; Pinter et al., 2017). To train such a model, it
7717 is only necessary to iterate over the vocabulary, and the not the corpus.

7718 **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,

$$\min_{\mathbf{U}} \sum_{j=1}^V \|\mathbf{u}_i - \hat{\mathbf{u}}_i\|_2 + \sum_{(i,j) \in \mathcal{L}} \beta_{ij} \|\mathbf{u}_i - \mathbf{u}_j\|_2, \quad [14.28]$$

7719 where $\hat{\mathbf{u}}_i$ is the pretrained embedding of word i , and $\mathcal{L} = \{(i,j)\}$ is a lexicon of word
 7720 relations. The hyperparameter β_{ij} controls the importance of adjacent words having
 7721 similar embeddings; Faruqui et al. (2015) set it to the inverse of the degree of word i ,
 7722 $\beta_{ij} = |\{j : (i,j) \in \mathcal{L}\}|^{-1}$. Retrofitting improves performance on a range of intrinsic evalua-
 7723 tions, and gives small improvements on an extrinsic document classification task.

7724 **14.8 Distributed representations of multiword units**

7725 Can distributed representations extend to phrases, sentences, paragraphs, and beyond?
 7726 Before exploring this possibility, recall the distinction between distributed and distri-
 7727 butional representations. Neural embeddings such as WORD2VEC are both distributed
 7728 (vector-based) and distributional (derived from counts of words in context). As we con-
 7729 sider larger units of text, the counts decrease: in the limit, a multi-paragraph span of text
 7730 would never appear twice, except by plagiarism. Thus, the meaning of a large span of
 7731 text cannot be determined from distributional statistics alone; it must be computed com-
 7732 positionally from smaller spans. But these considerations are orthogonal to the question
 7733 of whether distributed representations — dense numerical vectors — are sufficiently ex-
 7734 pressive to capture the meaning of phrases, sentences, and paragraphs.

7735 **14.8.1 Purely distributional methods**

7736 Some multiword phrases are non-compositional: the meaning of such phrases is not de-
 7737 rived from the meaning of the individual words using typical compositional semantics.
 7738 This includes proper nouns like *San Francisco* as well as idiomatic expressions like *kick*
 7739 *the bucket* (Baldwin and Kim, 2010). For these cases, purely distributional approaches
 7740 can work. A simple approach is to identify multiword units that appear together fre-
 7741 quently, and then treat these units as words, learning embeddings using a technique such
 7742 as WORD2VEC. The problem of identifying multiword units is sometimes called **colloca-**
 7743 **tion extraction**, and can be approached using metrics such as pointwise mutual informa-
 7744 tion: two-word units are extracted first, and then larger units are extracted. Mikolov et al.

7745 (2013) identify such units and then treat them as words when estimating skipgram em-
7746 beddings, showing that the resulting embeddings perform reasonably on a task of solving
7747 phrasal analogies, e.g. *New York : New York Times :: Baltimore : Baltimore Sun*.

7748 14.8.2 Distributional-compositional hybrids

7749 To move beyond short multiword phrases, composition is necessary. A simple but sur-
7750 prisingly powerful approach is to represent a sentence with the average of its word em-
7751 beddings (Mitchell and Lapata, 2010). This can be considered a hybrid of the distribu-
7752 tional and compositional approaches to semantics: the word embeddings are computed
7753 distributionally, and then the sentence representation is computed by composition.

7754 The WORD2VEC approach can be stretched considerably further, embedding entire
7755 sentences using a model similar to skipgrams, in the “skip-thought” model of Kiros et al.
7756 (2015). Each sentence is *encoded* into a vector using a recurrent neural network: the encod-
7757 ing of sentence t is set to the RNN hidden state at its final token, $h_{M_t}^{(t)}$. This vector is then
7758 a parameter in a *decoder* model that is used to generate the previous and subsequent sen-
7759 tences: the decoder is another recurrent neural network, which takes the encoding of the
7760 neighboring sentence as an additional parameter in its recurrent update. (This **encoder-**
7761 **decoder model** is discussed at length in chapter 18.) The encoder and decoder are trained
7762 simultaneously from a likelihood-based objective, and the trained encoder can be used to
7763 compute a distributed representation of any sentence. Skip-thought can also be viewed
7764 as a hybrid of distributional and compositional approaches: the vector representation of
7765 each sentence is computed compositionally from the representations of the individual
7766 words, but the training objective is distributional, based on sentence co-occurrence across
7767 a corpus.

7768 **Autoencoders** are a variant of encoder-decoder models in which the decoder is trained
7769 to produce the same text that was originally encoded, using only the distributed encod-
7770 ing vector (Li et al., 2015). The encoding acts as a bottleneck, so that generalization is
7771 necessary if the model is to successfully fit the training data. In **denoising autoencoders**,
7772 the input is a corrupted version of the original sentence, and the auto-encoder must re-
7773 construct the uncorrupted original (Vincent et al., 2010; Hill et al., 2016). By interpolating
7774 between distributed representations of two sentences, $\alpha \mathbf{u}_i + (1 - \alpha) \mathbf{u}_j$, it is possible to gen-
7775 erate sentences that combine aspects of the two inputs, as shown in Figure 14.6 (Bowman
7776 et al., 2016).

7777 Autoencoders can also be applied to longer texts, such as paragraphs and documents.
7778 This enables applications such as **question answering**, which can be performed by match-
7779 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)

7780 14.8.3 Supervised compositional methods

7781 Given a supervision signal, such as a label describing the sentiment or meaning of a sen-
 7782 tence, a wide range of compositional methods can be applied to compute a distributed
 7783 representation that then predicts the label. The simplest is to average the embeddings
 7784 of each word in the sentence, and pass this average through a feedforward neural net-
 7785 work (Iyyer et al., 2015). Convolutional and recurrent neural networks go further, with
 7786 the ability to effectively capturing multiword phenomena such as negation (Kalchbrenner
 7787 et al., 2014; Kim, 2014; Li et al., 2015; Tang et al., 2015). Another approach is to incorpo-
 7788 rate the syntactic structure of the sentence into a **recursive neural networks**, in which the
 7789 representation for each syntactic constituent is computed from the representations of its
 7790 children (Socher et al., 2012). However, in many cases, recurrent neural networks perform
 7791 as well or better than recursive networks (Li et al., 2015).

7792 Whether convolutional, recurrent, or recursive, a key question is whether supervised
 7793 sentence representations are task-specific, or whether a single supervised sentence repre-
 7794 sentation model can yield useful performance on other tasks. Wieting et al. (2015) train a
 7795 variety of sentence embedding models for the task of labeling pairs of sentences as **para-**
 7796 **phrases**. They show that the resulting sentence embeddings give good performance for
 7797 sentiment analysis. The **Stanford Natural Language Inference corpus** classifies sentence
 7798 pairs as **entailments** (the truth of sentence i implies the truth of sentence j), **contradictions**
 7799 (the truth of sentence i implies the falsity of sentence j), and neutral (i neither entails nor
 7800 contradicts j). Sentence embeddings trained on this dataset transfer to a wide range of
 7801 classification tasks (Conneau et al., 2017).

14.8.4 Hybrid distributed-symbolic representations

The power of distributed representations is in their generality: the distributed representation of a unit of text can serve as a summary of its meaning, and therefore as the input for downstream tasks such as classification, matching, and retrieval. For example, distributed sentence representations can be used to recognize the paraphrase relationship between closely related sentences like the following:

- (14.5) Donald thanked Vlad profusely.
- (14.6) Donald conveyed to Vlad his profound appreciation.
- (14.7) Vlad was showered with gratitude by Donald.

Symbolic representations are relatively brittle to this sort of variation, but are better suited to describe individual entities, the things that they do, and the things that are done to them. In examples (14.5)-(14.7), we not only know that somebody thanked someone else, but we can make a range of inferences about what has happened between the entities named *Donald* and *Vlad*. Because distributed representations do not treat entities symbolically, they lack the ability to reason about the roles played by entities across a sentence or larger discourse.⁵ A hybrid between distributed and symbolic representations might give the best of both worlds: robustness to the many different ways of describing the same event, plus the expressiveness to support inferences about entities and the roles that they play.

A “top-down” hybrid approach is to begin with logical semantics (of the sort described in the previous two chapters), and but replace the predefined lexicon with a set of distributional word clusters (Poon and Domingos, 2009; Lewis and Steedman, 2013). A “bottom-up” approach is to add minimal symbolic structure to existing distributed representations, such as vector representations for each entity (Ji and Eisenstein, 2015; Wiseman et al., 2016). This has been shown to improve performance on two problems that we will encounter in the following chapters: classification of **discourse relations** between adjacent sentences (chapter 16; Ji and Eisenstein, 2015), and **coreference resolution** of entity mentions (chapter 15; Wiseman et al., 2016; Ji et al., 2017). Research on hybrid semantic representations is still in an early stage, and future representations may deviate more boldly from existing symbolic and distributional approaches.

Additional resources

Turney and Pantel (2010) survey a number of facets of vector word representations, focusing 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!”

7835 similarity-based evaluations of word embeddings, and present a novel evaluation that
 7836 controls for word frequency. Baroni et al. (2014) address linguistic issues that arise in
 7837 attempts to combine distributed and compositional representations.

7838 In bilingual and multilingual distributed representations, embeddings are estimated
 7839 for translation pairs or tuples, such as (*dog, perro, chien*). These embeddings can improve
 7840 machine translation (Zou et al., 2013; Klementiev et al., 2012), transfer natural language
 7841 processing models across languages (Täckström et al., 2012), and make monolingual word
 7842 embeddings more accurate (Faruqui and Dyer, 2014). A typical approach is to learn a pro-
 7843 jection that maximizes the correlation of the distributed representations of each element
 7844 in a translation pair, which can be obtained from a bilingual dictionary. Distributed rep-
 7845 resentations can also be linked to perceptual information, such as image features. Bruni
 7846 et al. (2014) use textual descriptions of images to obtain visual contextual information for
 7847 various words, which supplements traditional distributional context. Image features can
 7848 also be inserted as contextual information in log bilinear language models (Kiros et al.,
 7849 2014), making it possible to automatically generate text descriptions of images.

7850 Exercises

- 7851 1. Prove that the sum of probabilities of paths through a hierarchical softmax tree is
 7852 equal to one.
- 7853 2. In skipgram word embeddings, the negative sampling objective can be written as,

$$\mathcal{L} = \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{C}} \text{count}(i, j) \psi(i, j), \quad [14.29]$$

7854 with $\psi(i, j)$ is defined in Equation 14.23.

7855 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 to this probability.

7856 Next, take the derivative of this expectation with respect to the score of a single word
 7857 context pair $\sigma(\mathbf{u}_i \cdot \mathbf{v}_j)$, and solve for the pointwise mutual information $\text{PMI}(i, j)$. You
 7858 should be able to show that at the optimum, the PMI is a simple function of $\sigma(\mathbf{u}_i \cdot \mathbf{v}_j)$
 7859 and the number of negative samples.

- 7860 3. * In Brown clustering, prove that the cluster merge that maximizes the average mu-
 7861 tual information (Equation 14.13) also maximizes the log-likelihood objective (Equa-
 7862 tion 14.12).
- 7863 4. A simple way to compute a distributed phrase representation is to add up the dis-
 7864 tributed representations of the words in the phrase. Consider a sentiment analysis
 7865 model in which the predicted sentiment is, $\psi(\mathbf{w}) = \boldsymbol{\theta} \cdot (\sum_{m=1}^M \mathbf{x}_m)$, where \mathbf{x}_m is

the vector representation of word m . Prove that in such a model, the following two inequalities cannot both hold:

$$\psi(\text{good}) > \psi(\text{not good}) \quad [14.30]$$

$$\psi(\text{bad}) < \psi(\text{not bad}). \quad [14.31]$$

Then construct a similar example pair for the case in which phrase representations are the *average* of the word representations.

5. Now let's consider a slight modification to the prediction model in the previous problem:

$$\psi(\mathbf{w}) = \boldsymbol{\theta} \cdot \text{ReLU}\left(\sum_{m=1}^M \mathbf{x}_m\right) \quad [14.32]$$

Show that in this case, it *is* possible to achieve the inequalities above. Your solution should provide the weights $\boldsymbol{\theta}$ and the embeddings \mathbf{x}_{good} , \mathbf{x}_{bad} , and \mathbf{x}_{not} .

- For the next two problems, download a set of pre-trained word embeddings, such as the WORD2VEC or polyglot embeddings.

6. Use cosine similarity to find the most similar words to: *dog*, *whale*, *before*, *however*, *fabricate*.

7. Use vector addition and subtraction to compute target vectors for the analogies below. After computing each target vector, find the top three candidates by cosine similarity.

- *dog:puppy :: cat: ?*
- *speak:speaker :: sing: ?*
- *France:French :: England: ?*
- *France:wine :: England: ?*

- The remaining problems will require you to build a classifier and test its properties. Pick a multi-class text classification dataset, such as RCV1⁶). Divide your data into training (60%), development (20%), and test sets (20%), if no such division already exists.

⁶http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm

- 7881 8. Train a convolutional neural network, with inputs set to pre-trained word embed-
7882 dings from the previous problem. Use a special, fine-tuned embedding for out-of-
7883 vocabulary words. Train until performance on the development set does not im-
7884 prove. You can also use the development set to tune the model architecture, such
7885 as the convolution width and depth. Report *F-MEASURE* and accuracy, as well as
7886 training time.
- 7887 9. Now modify your model from the previous problem to fine-tune the word embed-
7888 dings. Report *F-MEASURE*, accuracy, and training time.
- 7889 10. Try a simpler approach, in which word embeddings in the document are averaged,
7890 and then this average is passed through a feed-forward neural network. Again, use
7891 the development data to tune the model architecture. How close is the accuracy to
7892 the convolutional networks from the previous problems?

7893

Chapter 15

7894

Reference Resolution

7895 References are one of the most noticeable forms of linguistic ambiguity, afflicting not just
7896 automated natural language processing systems, but also fluent human readers. Warnings
7897 to avoid “ambiguous pronouns” are ubiquitous in manuals and tutorials on writing
7898 style. But referential ambiguity is not limited to pronouns, as shown in the text in Fig-
7899 ure 15.1. Each of the bracketed substrings refers to an entity that is introduced earlier
7900 in the passage. These references include the pronouns *he* and *his*, but also the shortened
7901 name *Cook*, and **nominals** such as *the firm* and *the firm’s biggest growth market*.

7902 **Reference resolution** subsumes several subtasks. This chapter will focus on **corefer-
7903 ence resolution**, which is the task of grouping spans of text that refer to a single underly-
7904 ing entity, or, in some cases, a single event: for example, the spans *Tim Cook*, *he*, and *Cook*
7905 are all **coreferent**. These individual spans are called **mentions**, because they mention an
7906 entity; the entity is sometimes called the **referent**. Each mention has a set of **antecedents**,
7907 which are preceding mentions that are coreferent; for the first mention of an entity, the an-
7908 tecedent set is empty. The task of **pronominal anaphora resolution** requires identifying
7909 only the antecedents of pronouns. In **entity linking**, references are resolved not to other
7910 spans of text, but to entities in a knowledge base. This task is discussed in chapter 17.

7911 Coreference resolution is a challenging problem for several reasons. Resolving differ-
7912 ent types of **referring expressions** requires different types of reasoning: the features and
7913 methods that are useful for resolving pronouns are different from those that are useful
7914 to resolve names and nominals. Coreference resolution involves not only linguistic rea-
7915 soning, but also world knowledge and pragmatics: you may not have known that China
7916 was Apple’s biggest growth market, but it is likely that you effortlessly resolved this ref-
7917 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.

resolution decisions are often entangled: each mention adds information about the entity, which affects other coreference decisions. This means that coreference resolution must be addressed as a structure prediction problem. But as we will see, there is no dynamic program that allows the space of coreference decisions to be searched efficiently.

15.1 Forms of referring expressions

There are three main forms of referring expressions — pronouns, names, and nominals.

15.1.1 Pronouns

Pronouns are a closed class of words that are used for references. A natural way to think about pronoun resolution is SMASH (Kehler, 2007):

- Search for candidate antecedents;
- Match against hard agreement constraints;
- And Select using Heuristics, which are “soft” constraints such as recency, syntactic prominence, and parallelism.

15.1.1.1 Search

In the search step, candidate antecedents are identified from the preceding text or speech.² 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.

7934 parsing the text to identify all such noun phrases.³ Filtering heuristics can help to prune
 7935 the search space to noun phrases that are likely to be coreferent (Lee et al., 2013; Durrett
 7936 and Klein, 2013). In nested noun phrases, mentions are generally considered to be the
 7937 largest unit with a given head word: thus, *Apple Inc. Chief Executive Tim Cook* would be
 7938 included as a mention, but *Tim Cook* would not, since they share the same head word,
 7939 *Cook*.

7940 15.1.1.2 Matching constraints for pronouns

7941 References and their antecedents must agree on semantic features such as number, person,
 7942 gender, and animacy. Consider the pronoun *he* in this passage from the running example:

7943 (15.2) Tim Cook has jetted in for talks with officials as [he] seeks to clear up a pile of
 7944 problems...

7945 The pronoun and possible antecedents have the following features:

- 7946 • *he*: singular, masculine, animate, third person
- 7947 • *officials*: plural, animate, third person
- 7948 • *talks*: plural, inanimate, third person
- 7949 • *Tim Cook*: singular, masculine, animate, third person

7950 The SMASH method searches backwards from *he*, discarding *officials* and *talks* because they
 7951 do not satisfy the agreements constraints.

7952 Another source of constraints comes from syntax — specifically, from the phrase struc-
 7953 ture trees discussed in chapter 10. Consider a parse tree in which both *x* and *y* are phrasal
 7954 constituents. The constituent *x* **c-commands** the constituent *y* iff the first branching node
 7955 above *x* also dominates *y*. For example, in Figure 15.2a, *Abigail* c-commands *her*, because
 7956 the first branching node above *Abigail*, *S*, also dominates *her*. Now, if *x* c-commands *y*,
 7957 **government and binding theory** (Chomsky, 1982) states that *y* can refer to *x* only if it is
 7958 a **reflexive pronoun** (e.g., *herself*). Furthermore, if *y* is a reflexive pronoun, then its an-
 7959 tecedent must c-command it. Thus, in Figure 15.2a, *her* cannot refer to *Abigail*; conversely,
 7960 if we replace *her* with *herself*, then the reflexive pronoun *must* refer to *Abigail*, since this is
 7961 the only candidate antecedent that c-commands it.

7962 Now consider the example shown in Figure 15.2b. Here, *Abigail* does not c-command
 7963 *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.

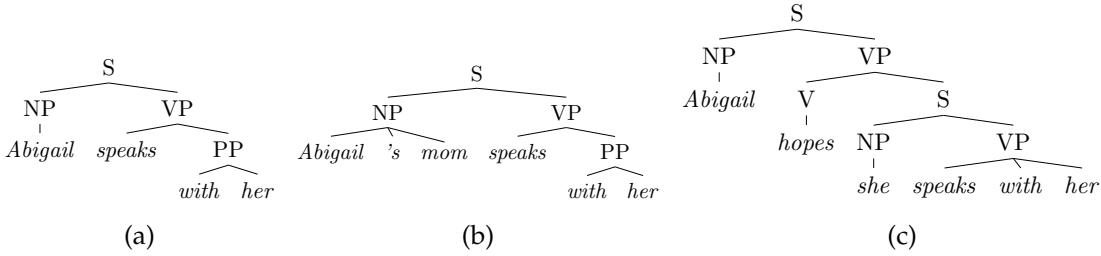


Figure 15.2: In (a), *Abigail* c-commands *her*; in (b), *Abigail* does not c-command *her*, but *Abigail's mom* does; in (c), the scope of *Abigail* is limited by the S non-terminal, so that *she* or *her* can bind to *Abigail*, but not both.

7964 *herself* in this context, unless we are talking about *Abigail*'s mom. However, *her* does not
 7965 have to refer to *Abigail*. Finally, Figure 15.2c shows how these constraints are limited.
 7966 In this case, the pronoun *she* can refer to *Abigail*, because the S non-terminal puts *Abigail*
 7967 outside the domain of *she*. Similarly, *her* can also refer to *Abigail*. But *she* and *her* cannot be
 7968 coreferent, because *she* c-commands *her*.

7969 15.1.1.3 Heuristics

7970 After applying constraints, heuristics are applied to select among the remaining candidates.
 7971 Recency is a particularly strong heuristic. All things equal, readers will prefer
 7972 the more recent referent for a given pronoun, particularly when comparing referents that
 7973 occur in different sentences. Jurafsky and Martin (2009) offer the following example:

- 7974 (15.3) The doctor found an old map in the captain's chest. Jim found an even older map
 7975 hidden on the shelf. [It] described an island.

7976 Readers are expected to prefer the older map as the referent for the pronoun *it*.

7977 However, subjects are often preferred over objects, and this can contradict the preference
 7978 for recency when two candidate referents are in the same sentence. For example,

- 7979 (15.4) Asha loaned Mei a book on Spanish. [She] is always trying to help people.

7980 Here, we may prefer to link *she* to *Asha* rather than *Mei*, because of *Asha*'s position in the
 7981 subject role of the preceding sentence. (Arguably, this preference would not be strong
 7982 enough to select *Asha* if the second sentence were *She is visiting Valencia next month*.)

7983 A third heuristic is parallelism:

- 7984 (15.5) Asha loaned Mei a book on Spanish. Olya loaned [her] a book on Portuguese.

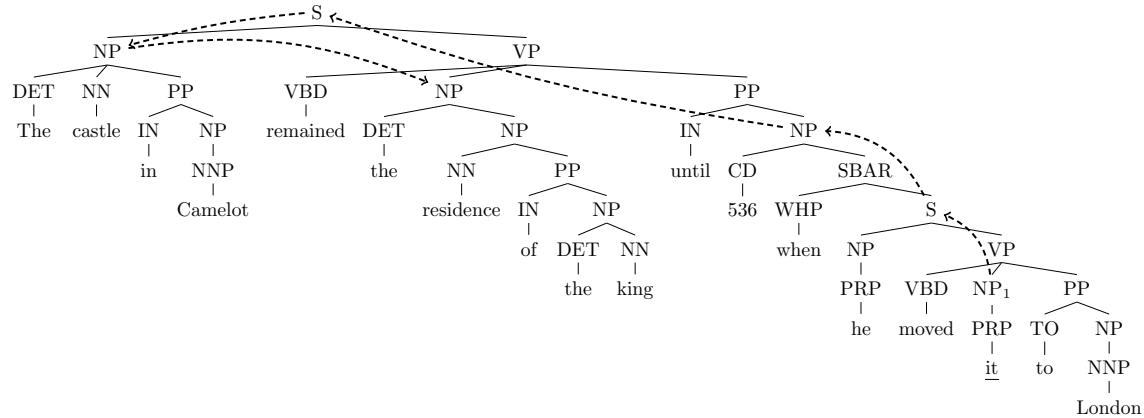


Figure 15.3: Left-to-right breadth-first tree traversal (Hobbs, 1978), indicating that the search for an antecedent for *it* (NP_1) would proceed in the following order: 536; *the castle in Camelot*; *the residence of the king*; *Camelot*; *the king*. Hobbs (1978) proposes semantic constraints to eliminate 536 and *the castle in Camelot* as candidates, since they are unlikely to be the direct object of the verb *move*.

7985 Here *Mei* is preferred as the referent for *her*, contradicting the preference for the subject
 7986 *Asha* in the preceding sentence.

7987 The recency and subject role heuristics can be unified by traversing the document in
 7988 a syntax-driven fashion (Hobbs, 1978): each preceding sentence is traversed breadth-first,
 7989 left-to-right (Figure 15.3). This heuristic successfully handles (15.4): *Asha* is preferred as
 7990 the referent for *she* because the subject NP is visited first. It also handles (15.3): the older
 7991 map is preferred as the referent for *it* because the more recent sentence is visited first. (An
 7992 alternative unification of recency and syntax is proposed by **centering theory** (Grosz et al.,
 7993 1995), which is discussed in detail in chapter 16.)

7994 In early work on reference resolution, the number of heuristics was small enough that
 7995 a set of numerical weights could be set by hand (Lappin and Leass, 1994). More recent
 7996 work uses machine learning to quantify the importance of each of these factors. However,
 7997 pronoun resolution cannot be completely solved by constraints and heuristics alone. This
 7998 is shown by the classic example pair (Winograd, 1972):

7999 (15.6) The [city council] denied [the protesters] a permit because [they] advocated / feared
 8000 violence.

8001 Without reasoning about the motivations of the city council and protesters, it is unlikely
 8002 that any system could correctly resolve both versions of this example.

8003 **15.1.1.4 Non-referential pronouns**

8004 While pronouns are generally used for reference, they need not refer to entities. The fol-
 8005 lowing examples show how pronouns can refer to propositions, events, and speech acts.

- 8006 (15.7) They told me that I was too ugly for show business, but I didn't believe [it].
 8007 (15.8) Asha saw Babak get angry, and I saw [it] too.
 8008 (15.9) Asha said she worked in security. I suppose [that]'s one way to put it.

8009 These forms of reference are generally not annotated in large-scale coreference resolution
 8010 datasets such as OntoNotes (Pradhan et al., 2011).

8011 Pronouns may also have **generic referents**:

- 8012 (15.10) A poor carpenter blames [her] tools.
 8013 (15.11) On the moon, [you] have to carry [your] own oxygen.
 8014 (15.12) Every farmer who owns a donkey beats [it]. (Geach, 1962)

8015 In the OntoNotes dataset, coreference is not annotated for generic referents, even in cases
 8016 like these examples, in which the same generic entity is mentioned multiple times.

8017 Some pronouns do not refer to anything at all:

- 8018 (15.13) *[It]'s raining.*
 [Il] pleut. (Fr)
 8019 (15.14) [It] 's money that she's really after.
 8020 (15.15) [It] is too bad that we have to work so hard.

8021 How can we automatically distinguish these usages of *it* from referential pronouns?
 8022 Consider the the difference between the following two examples (Bergsma et al., 2008):

- 8023 (15.16) You can make [it] in advance.
 8024 (15.17) You can make [it] in showbiz.

8025 In the second example, the pronoun *it* is non-referential. One way to see this is by substi-
 8026 tuting another pronoun, like *them*, into these examples:

- 8027 (15.18) You can make [them] in advance.
 8028 (15.19) ? You can make [them] in showbiz.

8029 The questionable grammaticality of the second example suggests that *it* is not referential.
 8030 Bergsma et al. (2008) operationalize this idea by comparing distributional statistics for the

8031 *n*-grams around the word *it*, testing how often other pronouns or nouns appear in the
8032 same context. In cases where nouns and other pronouns are infrequent, the *it* is unlikely
8033 to be referential.

8034 **15.1.2 Proper Nouns**

8035 If a proper noun is used as a referring expression, it often corefers with another proper
8036 noun, so that the coreference problem is simply to determine whether the two names
8037 match. Subsequent proper noun references often use a shortened form, as in the running
8038 example (Figure 15.1):

8039 (15.20) Apple Inc Chief Executive [Tim Cook] has jetted into China ... [Cook] is on his
8040 first business trip to the country ...

8041 A typical solution for proper noun coreference is to match the syntactic **head words**
8042 of the reference with the referent. In § 10.5.2, we saw that the head word of a phrase can
8043 be identified by applying head percolation rules to the phrasal parse tree; alternatively,
8044 the head can be identified as the root of the dependency subtree covering the name. For
8045 sequences of proper nouns, the head word will be the final token.

8046 There are a number of caveats to the practice of matching head words of proper nouns.

- 8047 • In the European tradition, family names tend to be more specific than given names,
8048 and family names usually come last. However, other traditions have other practices:
8049 for example, in Chinese names, the family name typically comes first; in Japanese,
8050 honorifics come after the name, as in *Nobu-San* (*Mr. Nobu*).
8051 • In organization names, the head word is often not the most informative, as in *Georgia*
8052 *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-
8053 fiers (Lee et al., 2011).
8054 • Proper nouns can be nested, as in [*the CEO of [Microsoft]*], resulting in head word
8055 match without coreference.

8056 Despite these difficulties, proper nouns are the easiest category of references to re-
8057 solve (Stoyanov et al., 2009). In machine learning systems, one solution is to include a
8058 range of matching features, including exact match, head match, and string inclusion. In
8059 addition to matching features, competitive systems (e.g., Bengtson and Roth, 2008) in-
8060 clude large lists, or **gazetteers**, of acronyms (e.g., *the National Basketball Association/NBA*),
8061 demonyms (e.g., *the Israelis/Israel*), and other aliases (e.g., *the Georgia Institute of Technol-
8062 ogy/Georgia Tech*).

8064 **15.1.3 Nominals**

8065 In coreference resolution, noun phrases that are neither pronouns nor proper nouns are
 8066 referred to as **nominals**. In the running example (Figure 15.1), nominal references include:
 8067 *the firm (Apple Inc); the firm's biggest growth market (China); and the country (China)*.

8068 Nominals are especially difficult to resolve (Denis and Baldridge, 2007; Durrett and
 8069 Klein, 2013), and the examples above suggest why this may be the case: world knowledge
 8070 is required to identify *Apple Inc* as a *firm*, and *China* as a *growth market*. Other difficult
 8071 examples include the use of colloquial expressions, such as coreference between *Clinton*
 8072 *campaign officials* and *the Clinton camp* (Soon et al., 2001).

8073 **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:

$$c_1 = \{Apple\ Inc_{1:2}, the\ firm_{27:28}\} \quad [15.1]$$

$$c_2 = \{Apple\ Inc\ Chief\ Executive\ Tim\ Cook_{1:6}, he_{17}, Cook_{33}, his_{36}\} \quad [15.2]$$

$$c_3 = \{China_{10}, the\ firm\ 's\ biggest\ growth\ market_{27:32}, the\ country_{40:41}\} \quad [15.3]$$

8074 Each row specifies the token spans that mention an entity. (“Singleton” entities, which are
 8075 mentioned only once (e.g., *talks, government officials*), are excluded from the annotations.)
 8076 Equivalently, if given a set of M mentions, $\{m_i\}_{i=1}^M$, each mention i can be assigned to a
 8077 cluster z_i , where $z_i = z_j$ if i and j are coreferent. The cluster assignments z are invariant
 8078 under permutation. The unique clustering associated with the assignment z is written
 8079 $c(z)$.

8080 Coreference resolution can thus be viewed as a structure prediction problem, involving
 8081 two subtasks: identifying which spans of text mention entities, and then clustering
 8082 those spans.

8083 **Mention identification** The task of identifying mention spans for coreference resolution
 8084 is often performed by applying a set of heuristics to the phrase structure parse of each
 8085 sentence. A typical approach is to start with all noun phrases and named entities, and
 8086 then apply filtering rules to remove nested noun phrases with the same head (e.g., [*Apple*
 8087 *CEO [Tim Cook]*]), numeric entities (e.g., [*100 miles*], [*97%*]), non-referential *it*, etc (Lee

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.

et al., 2013; Durrett and Klein, 2013). In general, these deterministic approaches err in favor of recall, since the mention clustering component can choose to ignore false positive mentions, but cannot recover from false negatives. An alternative is to consider all spans (up to some finite length) as candidate mentions, performing mention identification and clustering jointly (Daumé III and Marcu, 2005; Lee et al., 2017).

Mention clustering The subtask of mention clustering will be the focus of the remainder of this chapter. There are two main classes of models. In *mention-based models*, the scoring function for a coreference clustering decomposes over pairs of mentions. These pairwise decisions are then aggregated, using a clustering heuristic. Mention-based coreference clustering can be treated as a fairly direct application of supervised classification or ranking. However, the mention-pair locality assumption can result in incoherent clusters, like $\{\text{Hillary Clinton} \leftarrow \text{Clinton} \leftarrow \text{Mr Clinton}\}$, in which the pairwise links score well, but the overall result is unsatisfactory. *Entity-based models* address this issue by scoring entities holistically. This can make inference more difficult, since the number of possible entity groupings is exponential in the number of mentions.

15.2.1 Mention-pair models

In the **mention-pair model**, a binary label $y_{i,j} \in \{0, 1\}$ is assigned to each pair of mentions (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 mention *he* in Figure 15.1 is preceded by five other mentions: (1) *Apple Inc*; (2) *Apple Inc Chief Executive Tim Cook*; (3) *China*; (4) *talks*; (5) *government officials*. The correct mention 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, such as mention 3 in the example, then $y_{i,j} = 0$ for all i . The same is true for “mentions” that do not refer to any entity, such as non-referential pronouns. If mention j refers to an entity that has been mentioned more than once, then $y_{i,j} = 1$ for all $i < j$ that mention the referent.

By transforming coreference into a set of binary labeling problems, the mention-pair model makes it possible to apply an off-the-shelf binary classifier (Soon et al., 2001). This classifier is applied to each mention j independently, searching backwards from j until finding an antecedent i which corefers with j with high confidence. After identifying a single **antecedent**, the remaining mention pair labels can be computed by transitivity: if $y_{i,j} = 1$ and $y_{j,k} = 1$, then $y_{i,k} = 1$.

Since the ground truth annotations give entity chains c but not individual mention-pair labels y , an additional heuristic must be employed to convert the labeled data into training examples for classification. A typical approach is to generate at most one positive labeled instance $y_{a_j,j} = 1$ for mention j , where a_j is the index of the most recent antecedent, $a_j = \max\{i : i < j \wedge z_i = z_j\}$. Negative labeled instances are generated for all for all $i \in \{a_j + 1, \dots, j\}$. In the running example, the most recent antecedent of the

8125 pronoun *he* is $a_6 = 2$, so the training data would be $y_{2,6} = 1$ and $y_{3,6} = y_{4,6} = y_{5,6} = 0$.
 8126 The variable $y_{1,6}$ is not part of the training data, because the first mention appears before
 8127 the true antecedent $a_6 = 2$.

8128 **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 ,

$$\hat{a}_i = \operatorname{argmax}_{a \in \{\epsilon, 1, 2, \dots, i-1\}} \psi_M(a, i), \quad [15.4]$$

8129 where $\psi_M(a, i)$ is a score for the mention pair (a, i) . If $a = \epsilon$, then mention i does not refer
 8130 to any previously-introduced entity — it is not **anaphoric**. Mention-ranking is similar to
 8131 the mention-pair model, but all candidates are considered simultaneously, and at most
 8132 a single antecedent is selected. The mention-ranking model explicitly accounts for the
 8133 possibility that mention i is not anaphoric, through the score $\psi_M(\epsilon, i)$. The determination
 8134 of anaphoricity can be made by a special classifier in a preprocessing step, so that non- ϵ
 8135 antecedents are identified only for spans that are determined to be anaphoric (Denis and
 8136 Baldridge, 2008).

8137 As a learning problem, ranking can be trained using the same objectives as in dis-
 8138 criminative classification. For each mention i , we can define a gold antecedent a_i^* , and an
 8139 associated loss, such as the hinge loss, $\ell_i = (1 - \psi_M(a_i^*, i) + \psi_M(\hat{a}, i))_+$ or the negative
 8140 log-likelihood, $\ell_i = -\log p(a_i^* | i; \theta)$. (For more on learning to rank, see § 17.1.1.) But as
 8141 with the mention-pair model, there is a mismatch between the labeled data, which comes
 8142 in the form of mention sets, and the desired supervision, which would indicate the spe-
 8143 cific antecedent of each mention. The antecedent variables $\{a_i\}_{i=1}^M$ relate to the mention
 8144 sets in a many-to-one mapping: each set of antecedents induces a single clustering, but a
 8145 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:

$$p(c | \mathbf{m}) = \sum_{\mathbf{a} \in \mathcal{A}(c)} p(\mathbf{a} | \mathbf{m}) = \sum_{\mathbf{a} \in \mathcal{A}(c)} \prod_{i=1}^M p(a_i | i, \mathbf{m}) \quad [15.8]$$

$$p(a_i | i, \mathbf{m}) = \frac{\exp(\psi_M(a_i, i))}{\sum_{a' \in \{\epsilon, 1, 2, \dots, i-1\}} \exp(\psi_M(a', i))}. \quad [15.9]$$

- 8146 This objective rewards models that assign high scores for all valid antecedent structures.
 8147 In the running example, this would correspond to summing the probabilities of the two
 8148 valid antecedents for *Cook, he* and *Apple Inc Chief Executive Tim Cook*. In one of the exer-
 8149 cises, you will compute the number of valid antecedent structures for a given clustering.

8150 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:

$$m_1 = \text{Hillary Clinton} \quad [15.10]$$

$$m_2 = \text{Clinton} \quad [15.11]$$

$$m_3 = \text{Bill Clinton} \quad [15.12]$$

- 8151 A mention-pair system might predict $\hat{y}_{1,2} = 1, \hat{y}_{2,3} = 1, \hat{y}_{1,3} = 0$. Similarly, a mention-
 8152 ranking system might choose $\hat{a}_2 = 1$ and $\hat{a}_3 = 2$. Logically, if mentions 1 and 3 are both
 8153 coreferent with mention 2, then all three mentions must refer to the same entity. This
 8154 constraint is known as **transitive closure**.

8155 Transitive closure can be applied *post hoc*, revising the independent mention-pair or
 8156 mention-ranking decisions. However, there are many possible ways to enforce transitive
 8157 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 docu-
 8158 ments with many mentions, there may be many violations of transitive closure, and many
 8159 possible fixes. Transitive closure can be enforced by always adding edges, so that $\hat{y}_{1,3} = 1$
 8160 is preferred (e.g., Soon et al., 2001), but this can result in overclustering, with too many
 8161 mentions grouped into too few entities.

Mention-pair coreference resolution can be viewed as a constrained optimization prob-
 lem,

$$\max_{\mathbf{y} \in \{0,1\}^M} \sum_{j=1}^M \sum_{i=1}^j \psi_M(i, j) \times y_{i,j}$$

s.t. $y_{i,j} + y_{j,k} - 1 \leq y_{i,k}, \quad \forall i < j < k,$

8162 with the constraint enforcing transitive closure. This constrained optimization problem
 8163 is equivalent to graph partitioning with positive and negative edge weights: construct a
 8164 graph where the nodes are mentions, and the edges are the pairwise scores $\psi_M(i, j)$; the
 8165 goal is to partition the graph so as to maximize the sum of the edge weights between all
 8166 nodes within the same partition (McCallum and Wellner, 2004). This problem is NP-hard,
 8167 motivating approximations such as correlation clustering (Bansal et al., 2004) and **integer**
 8168 **linear programming** (Klenner, 2007; Finkel and Manning, 2008, also see § 13.2.2).

8169 15.2.4 Entity-based models

A weakness of mention-based models is that they treat coreference resolution as a classifi-
 cation 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 consis-
 tent. Coreference resolution can then be viewed as the following optimization,

$$\max_{\mathbf{z}} \sum_{e=1} \psi_E(\{i : z_i = e\}), \tag{15.13}$$

8170 where z_i indicates the entity referenced by mention i , and $\psi_E(\{i : z_i = e\})$ is a scoring
 8171 function applied to all mentions i that are assigned to entity e .

Entity-based coreference resolution is conceptually similar to the unsupervised clus-
 tering problems encountered in chapter 5: the goal is to obtain clusters of mentions that
 are internally coherent. The number of possible clusterings of n items is the **Bell number**,

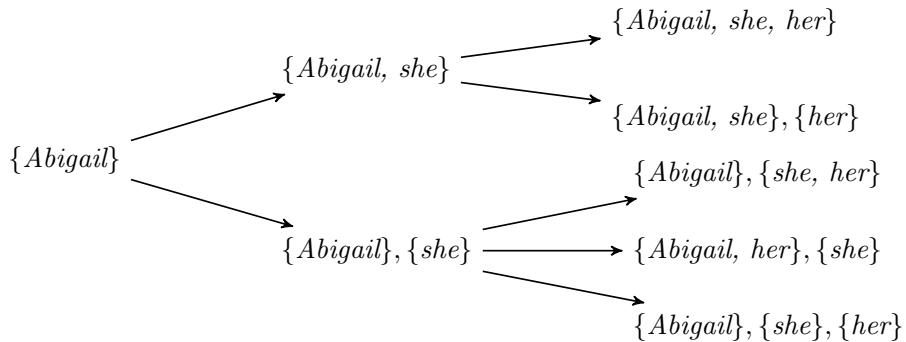


Figure 15.4: The Bell Tree for the sentence *Abigail hopes she speaks with her*. Which paths are excluded by the syntactic constraints mentioned in § 15.1.1?

which is defined by the following recurrence (Bell, 1934; Luo et al., 2004),

$$B_n = \sum_{k=0}^{n-1} B_k \binom{n-1}{k} B_0 = \quad B_1 = 1. \quad [15.14]$$

8172 This recurrence is illustrated by the Bell tree, which is applied to a short coreference prob-
 8173 lem in Figure 15.4. The Bell number B_n grows exponentially with n , making exhaustive
 8174 search of the space of clusterings impossible. For this reason, entity-based coreference
 8175 resolution typically involves incremental search, in which clustering decisions are based
 8176 on local evidence, in the hope of approximately optimizing the full objective in Equa-
 8177 tion 15.13. This approach is sometimes called **cluster ranking**, in contrast to mention
 8178 ranking.

8179 ***Generative models of coreference** Entity-based coreference can be approached through
 8180 probabilistic **generative models**, in which the mentions in the document are conditioned
 8181 on a set of latent entities (Haghghi and Klein, 2007, 2010). An advantage of these meth-
 8182 ods is that they can be learned from unlabeled data (Poon and Domingos, 2008, e.g.); a
 8183 disadvantage is that probabilistic inference is required not just for learning, but also for
 8184 prediction. Furthermore, generative models require independence assumptions that are
 8185 difficult to apply in coreference resolution, where the diverse and heterogeneous features
 8186 do not admit an easy decomposition into mutually independent subsets.

8187 15.2.4.1 Incremental cluster ranking

8188 The SMASH method (§ 15.1.1) can be extended to entity-based coreference resolution by
 8189 building up coreference clusters while moving through the document (Cardie and Wagstaff,
 8190 1999). At each mention, the algorithm iterates backwards through possible antecedent

8191 clusters; but unlike SMASH, a cluster is selected only if *all* members of its cluster are compatible
 8192 with the current mention. As mentions are added to a cluster, so are their features
 8193 (e.g., gender, number, animacy). In this way, incoherent chains like *{Hillary Clinton, Clinton, Bill Clinton}*
 8194 can be avoided. However, an incorrect assignment early in the document — a **search error**
 8195 — might lead to a cascade of errors later on.

8196 More sophisticated search strategies can help to ameliorate the risk of search errors.
 8197 One approach is **beam search** (§ 11.3), in which a set of hypotheses is maintained through-
 8198 out search. Each hypothesis represents a path through the Bell tree (Figure 15.4). Hy-
 8199 potheses are “expanded” either by adding the next mention to an existing cluster, or by
 8200 starting a new cluster. Each expansion receives a score, based on Equation 15.13, and the
 8201 top K hypotheses are kept on the beam as the algorithm moves to the next step.

8202 Incremental cluster ranking can be made more accurate by performing multiple passes
 8203 over the document, applying rules (or “sieves”) with increasing recall and decreasing
 8204 precision at each pass (Lee et al., 2013). In the early passes, coreference links are pro-
 8205 posed only between mentions that are highly likely to corefer (e.g., exact string match
 8206 for full names and nominals). Information can then be shared among these mentions,
 8207 so that when more permissive matching rules are applied later, agreement is preserved
 8208 across the entire cluster. For example, in the case of *{Hillary Clinton, Clinton, she}*, the
 8209 name-matching sieve would link *Clinton* and *Hillary Clinton*, and the pronoun-matching
 8210 sieve would then link *she* to the combined cluster. A deterministic multi-pass system
 8211 won nearly every track of the 2011 CoNLL shared task on coreference resolution (Prad-
 8212 han et al., 2011). Given the dominance of machine learning in virtually all other areas
 8213 of natural language processing — and more than fifteen years of prior work on machine
 8214 learning for coreference — this was a surprising result, even if learning-based methods
 8215 have subsequently regained the upper hand (e.g., Lee et al., 2017, the state-of-the-art at
 8216 the time of this writing).

8217 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,

$$\psi_E(\{i : z_i = e\}) = \sum_{i:z_i=e} \theta \cdot f(i, \{j : j < i \wedge z_j = e\}), \quad [15.15]$$

8218 where the score for each cluster is computed as the sum of scores of all mentions that are
 8219 linked into the cluster, and $f(i, \emptyset)$ is a set of features for the non-anaphoric mention that
 8220 initiates the cluster.

8221 Using Figure 15.4 as an example, suppose that the ground truth is,

$$\mathbf{c}^* = \{\text{Abigail}, \text{her}\}, \{\text{she}\}, \quad [15.16]$$

8222 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:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{f}(\mathbf{c}^*) - \mathbf{f}(\hat{\mathbf{c}}) \quad [15.18]$$

$$= \boldsymbol{\theta} + (\mathbf{f}(\text{Abigail}, \emptyset) + \mathbf{f}(\text{she}, \emptyset)) - (\mathbf{f}(\text{Abigail}, \emptyset) + \mathbf{f}(\text{she}, \{\text{Abigail}\})) \quad [15.19]$$

$$= \boldsymbol{\theta} + \mathbf{f}(\text{she}, \emptyset) - \mathbf{f}(\text{she}, \{\text{Abigail}\}). \quad [15.20]$$

8223 This style of incremental update can also be applied to a margin loss between the gold
 8224 clustering and the top clustering on the beam. By backpropagating from this loss, it is also
 8225 possible to train a more complicated scoring function, such as a neural network in which
 8226 the score for each entity is a function of embeddings for the entity mentions (Wiseman
 8227 et al., 2015).

8228 15.2.4.3 Reinforcement learning

8229 **Reinforcement learning** is a topic worthy of a textbook of its own (Sutton and Barto,
 8230 1998),⁵ so this section will provide only a very brief overview, in the context of coreference
 8231 resolution. A stochastic **policy** assigns a probability to each possible **action**, conditional
 8232 on the context. The goal is to learn a policy that achieves a high expected reward, or
 8233 equivalently, a low expected cost.

8234 In incremental cluster ranking, a complete clustering on M mentions can be produced
 8235 by a sequence of M actions, in which the action z_i either merges mention i with an existing
 8236 cluster or begins a new cluster. We can therefore create a stochastic policy using the cluster
 8237 scores (Clark and Manning, 2016),

$$\Pr(z_i = e; \boldsymbol{\theta}) = \frac{\exp \psi_E(i \cup \{j : z_j = e\}; \boldsymbol{\theta})}{\sum_{e'} \exp \psi_E(i \cup \{j : z_j = e'\}; \boldsymbol{\theta})}, \quad [15.21]$$

8238 where $\psi_E(i \cup \{j : z_j = e\}; \boldsymbol{\theta})$ is the score under parameters $\boldsymbol{\theta}$ for assigning mention i to
 8239 cluster e . This score can be an arbitrary function of the mention i , the cluster e and its
 8240 (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).

8241 If a policy assigns probability $p(c; \theta)$ to clustering c , then its expected loss is,

$$L(\theta) = \sum_{c \in \mathcal{C}(m)} p_\theta(c) \times \ell(c), \quad [15.22]$$

8242 where $\mathcal{C}(m)$ is the set of possible clusterings for mentions m . The loss $\ell(c)$ can be based on
 8243 any arbitrary scoring function, including the complex evaluation metrics used in corefer-
 8244 ence resolution (see § 15.4). This is an advantage of reinforcement learning, which can be
 8245 trained directly on the evaluation metric — unlike traditional supervised learning, which
 8246 requires a loss function that is differentiable and decomposable across individual deci-
 8247 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),

$$\frac{\partial}{\partial \theta} L(\theta) = E_{z \sim \mathcal{Z}(m)} \ell(c(z)) \sum_{i=1}^M \frac{\partial}{\partial \theta} \log p(z_i | z_{1:i-1}, m) \quad [15.23]$$

$$\approx \frac{1}{K} \sum_{k=1}^K \ell(c(z^{(k)})) \sum_{i=1}^M \frac{\partial}{\partial \theta} \log p(z_i^{(k)} | z_{1:i-1}^{(k)}, m), \quad [15.24]$$

8248 where each action sequence $z^{(k)}$ is sampled from the current policy. Unlike the incremen-
 8249 tal perceptron, an update is not made until the complete action sequence is available.

8250 15.2.4.4 Learning to search

8251 Policy gradient can suffer from high variance: while the average loss over K samples is
 8252 asymptotically equal to the expected reward of a given policy, this estimate may not be
 8253 accurate unless K is very large. This can make it difficult to allocate credit and blame to
 8254 individual actions. In **learning to search**, this problem is addressed through the addition
 8255 of an **oracle** policy, which is known to receive zero or small loss. The oracle policy can be
 8256 used in two ways:

- 8257 • The oracle can be used to generate partial hypotheses that are likely to score well,
 8258 by generating i actions from the initial state. These partial hypotheses are then used
 8259 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  $\frac{\partial}{\partial \theta} L(\theta)$ 

```

- 8260 • The oracle can be used to compute the minimum possible loss from a given state, by
 8261 generating $M - i$ actions from the current state until completion. This is known as
 8262 **roll-out**.

8263 The oracle can be combined with the existing policy during both roll-in and roll-out, sam-
 8264 pling actions from each policy (Daumé III et al., 2009). One approach is to gradually
 8265 decrease the number of actions drawn from the oracle over the course of learning (Ross
 8266 et al., 2011).

8267 In the context of entity-based coreference resolution, Clark and Manning (2016) use
 8268 the learned policy for roll-in and the oracle policy for roll-out. Algorithm 18 shows how
 8269 the gradients on the policy weights are computed in this case. In this application, the
 8270 oracle is “noisy”, because it selects the action that minimizes only the *local* loss — the
 8271 accuracy of the coreference clustering up to mention i — rather than identifying the action
 8272 sequence that will lead to the best final coreference clustering on the entire document.
 8273 When learning from noisy oracles, it can be helpful to mix in actions from the current
 8274 policy with the oracle during roll-out (Chang et al., 2015).

8275 **15.3 Representations for coreference resolution**

8276 Historically, coreference resolution has employed an array of hand-engineered features
 8277 to capture the linguistic constraints and preferences described in § 15.1 (Soon et al., 2001).
 8278 Later work has documented the utility of lexical and bilexical features on mention pairs (Björkelund
 8279 and Nugues, 2011; Durrett and Klein, 2013). The most recent and successful methods re-
 8280 place many (but not all) of these features with distributed representations of mentions
 8281 and entities (Wiseman et al., 2015; Clark and Manning, 2016; Lee et al., 2017).

8282 **15.3.1 Features**

8283 Coreference features generally rely on a preprocessing pipeline to provide part-of-speech
 8284 tags and phrase structure parses. This pipeline makes it possible to design features that
 8285 capture many of the phenomena from § 15.1, and is also necessary for typical approaches
 8286 to mention identification. However, the pipeline may introduce errors that propagate
 8287 to the downstream coreference clustering system. Furthermore, the existence of such
 8288 a pipeline presupposes resources such as treebanks, which do not exist for many lan-
 8289 guages.⁶

8290 **15.3.1.1 Mention features**

8291 Features of individual mentions can help to predict anaphoricity. In systems where men-
 8292 tion detection is performed jointly with coreference resolution, these features can also
 8293 predict whether a span of text is likely to be a mention. For mention i , typical features
 8294 include:

8295 **Mention type.** Each span can be identified as a pronoun, name, or nominal, using the
 8296 part-of-speech of the head word of the mention: both the Penn Treebank and Uni-
 8297 versal Dependencies tagsets (§ 8.1.1) include tags for pronouns and proper nouns,
 8298 and all other heads can be marked as nominals (Haghghi and Klein, 2009).

8299 **Mention width.** The number of tokens in a mention is a rough predictor of its anaphor-
 8300 icity, with longer mentions being less likely to refer back to previously-defined enti-
 8301 ties.

8302 **Lexical features.** The first, last, and head words can help to predict anaphoricity; they are
 8303 also useful in conjunction with features such as mention type and part-of-speech,
 8304 providing a rough measure of agreement (Björkelund and Nugues, 2011). The num-
 8305 ber of lexical features can be very large, so it can be helpful to select only frequently-
 8306 occurring features (Durrett and Klein, 2013).

8307 **Morphosyntactic features.** These features include the part-of-speech, number, gender,
 8308 and dependency ancestors.

8309 The features for mention i and candidate antecedent a can be conjoined, producing
 8310 joint features that can help to assess the compatibility of the two mentions. For example,
 8311 Durrett and Klein (2013) conjoin each feature with the mention types of the anaphora
 8312 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>

8313 documents from various genres. By conjoining the genre with other features, it is possible
8314 to learn genre-specific feature weights.

8315 **15.3.1.2 Mention-pair features**

8316 For any pair of mentions i and j , typical features include:

8317 **Distance.** The number of intervening tokens, mentions, and sentences between i and j
8318 can all be used as distance features. These distances can be computed on the surface
8319 text, or on a transformed representation reflecting the breadth-first tree traversal
8320 (Figure 15.3). Rather than using the distances directly, they are typically binned,
8321 creating binary features.

8322 **String match.** A variety of string match features can be employed: exact match, suffix
8323 match, head match, and more complex matching rules that disregard irrelevant
8324 modifiers (Soon et al., 2001).

8325 **Compatibility.** Building on the model, features can measure the anaphor and antecedent
8326 agree with respect to morphosyntactic attributes such as gender, number, and ani-
8327 macy.

8328 **Nesting.** If one mention is nested inside another (e.g., *[The President of [France]]*), they
8329 generally cannot corefer.

8330 **Same speaker.** For documents with quotations, such as news articles, personal pronouns
8331 can be resolved only by determining the speaker for each mention (Lee et al., 2013).
8332 Coreference is also more likely between mentions from the same speaker.

8333 **Gazetteers.** These features indicate that the anaphor and candidate antecedent appear in
8334 a gazetteer of acronyms (e.g., *USA/United States*, *GATech/Georgia Tech*), demonymns
8335 (e.g., *Israel/Israeli*), or other aliases (e.g., *Knickerbockers/New York Knicks*).

8336 **Lexical semantics.** These features use a lexical resource such as WordNet to determine
8337 whether the head words of the mentions are related through synonymy, antonymy,
8338 and hypernymy (§ 4.2).

8339 **Dependency paths.** The dependency path between the anaphor and candidate antecedent
8340 can help to determine whether the pair can corefer, under the government and bind-
8341 ing constraints described in § 15.1.1.

8342 Comprehensive lists of mention-pair features are offered by Bengtson and Roth (2008) and
8343 Rahman and Ng (2011). Neural network approaches use far fewer mention-pair features:
8344 for example, Lee et al. (2017) include only speaker, genre, distance, and mention width
8345 features.

8346 **Semantics** In many cases, coreference seems to require knowledge and semantic in-
 8347 ferences, as in the running example, where we link *China* with a *country* and a *growth*
 8348 *market*. Some of this information can be gleaned from WordNet, which defines a graph
 8349 over **synsets** (see § 4.2). For example, one of the synsets of *China* is an instance of an
 8350 *Asian_nation#1*, which in turn is a hyponym of *country#2*, a synset that includes
 8351 *country*.⁷ Such paths can be used to measure the similarity between concepts (Pedersen
 8352 et al., 2004), and this similarity can be incorporated into coreference resolution as a fea-
 8353 ture (Ponzetto and Strube, 2006). Similar ideas can be applied to knowledge graphs in-
 8354 duced from Wikipedia (Ponzetto and Strube, 2007). But while such approaches improve
 8355 relatively simple classification-based systems, they have proven less useful when added
 8356 to the current generation of techniques.⁸ For example, Durrett and Klein (2013) employ
 8357 a range of semantics-based features — WordNet synonymy and hypernymy relations on
 8358 head words, named entity types (e.g., person, organization), and unsupervised clustering
 8359 over nominal heads — but find that these features give minimal improvement over a
 8360 baseline system using surface features.

8361 15.3.1.3 Entity features

8362 Many of the features for entity-mention coreference are generated by aggregating mention-
 8363 pair features over all mentions in the candidate entity (Culotta et al., 2007; Rahman and
 8364 Ng, 2011). Specifically, for each binary mention-pair feature $f(i, j)$, we compute the fol-
 8365 lowing entity-mention features for mention i and entity $e = \{j : j < i \wedge z_j = e\}$.

- 8366 • ALL-TRUE: Feature $f(i, j)$ holds for all mentions $j \in e$.
- 8367 • MOST-TRUE: Feature $f(i, j)$ holds for at least half and fewer than all mentions $j \in e$.
- 8368 • MOST-FALSE: Feature $f(i, j)$ holds for at least one and fewer than half of all men-
 8369 tions $j \in e$.
- 8370 • NONE: Feature $f(i, j)$ does not hold for any mention $j \in e$.

8371 For scalar mention-pair features (e.g., distance features), aggregation can be performed by
 8372 computing the minimum, maximum, and median values across all mentions in the cluster.
 8373 Additional entity-mention features include the number of mentions currently clustered in
 8374 the entity, and ALL-X and MOST-X features for each mention type.

8375 15.3.2 Distributed representations of mentions and entities

8376 Recent work has emphasized distributed representations of both mentions and entities.
 8377 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.

8378 mantic compatibility underlying nominal coreference, helping with difficult cases like
 8379 (*Apple, the firm*) and (*China, the firm's biggest growth market*). Furthermore, a distributed
 8380 representation of entities can be trained to capture semantic features that are added by
 8381 each mention.

8382 **15.3.2.1 Mention embeddings**

8383 Entity mentions can be embedded into a vector space, providing the base layer for neural
 8384 networks that score coreference decisions (Wiseman et al., 2015).

8385 **Constructing the mention embedding** Various approaches for embedding multiword
 8386 units can be applied (see § 14.8). Figure 15.5 shows a recurrent neural network approach,
 8387 which begins by running a bidirectional LSTM over the entire text, obtaining hidden states
 8388 from the left-to-right and right-to-left passes, $\mathbf{h}_m = [\overleftarrow{\mathbf{h}}_m; \overrightarrow{\mathbf{h}}_m]$. Each candidate mention
 8389 span (s, t) is then represented by the vertical concatenation of four vectors:

$$\mathbf{u}^{(s,t)} = [\mathbf{u}_{\text{first}}^{(s,t)}; \mathbf{u}_{\text{last}}^{(s,t)}; \mathbf{u}_{\text{head}}^{(s,t)}; \phi^{(s,t)}], \quad [15.25]$$

8390 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
 8391 embedding of the last word, $\mathbf{u}_{\text{head}}^{(s,t)}$ is the embedding of the “head” word, and $\phi^{(s,t)}$ is a
 8392 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**:

$$\tilde{\alpha}_m = \theta_\alpha \cdot \mathbf{h}_m \quad [15.26]$$

$$\mathbf{a}^{(s,t)} = \text{SoftMax}([\tilde{\alpha}_{s+1}, \tilde{\alpha}_{s+2}, \dots, \tilde{\alpha}_t]) \quad [15.27]$$

$$\mathbf{u}_{\text{head}}^{(s,t)} = \sum_{m=s+1}^t a_m^{(s,t)} \mathbf{h}_m. \quad [15.28]$$

8393 Each token m gets a scalar score $\tilde{\alpha}_m = \theta_\alpha \cdot \mathbf{h}_m$, which is the dot product of the LSTM
 8394 hidden state \mathbf{h}_m and a vector of weights θ_α . The vector of scores for tokens in the span
 8395 $m \in \{s + 1, s + 2, \dots, t\}$ is then passed through a softmax layer, yielding a vector $\mathbf{a}^{(s,t)}$
 8396 that allocates one unit of attention across the span. This eliminates the need for syntactic
 8397 parsing to recover the head word; instead, the model learns to identify the most important
 8398 words in each span. Attention mechanisms were introduced in neural machine transla-
 8399 tion (Bahdanau et al., 2014), and are described in more detail in § 18.3.1.

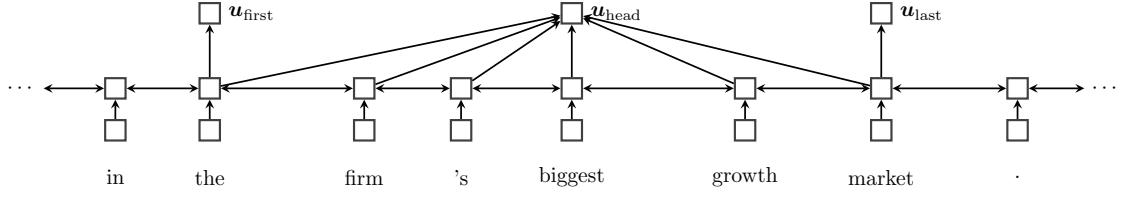


Figure 15.5: A bidirectional recurrent model of mention embeddings. The mention is represented by its first word, its last word, and an estimate of its head word, which is computed from a weighted average (Lee et al., 2017).

Using mention embeddings Given a set of mention embeddings, each mention i and candidate antecedent a is scored as,

$$\psi(a, i) = \psi_S(a) + \psi_S(i) + \psi_M(a, i) \quad [15.29]$$

$$\psi_S(a) = \text{FeedForward}_S(\mathbf{u}^{(a)}) \quad [15.30]$$

$$\psi_S(i) = \text{FeedForward}_S(\mathbf{u}^{(i)}) \quad [15.31]$$

$$\psi_M(a, i) = \text{FeedForward}_M([\mathbf{u}^{(a)}; \mathbf{u}^{(i)}; \mathbf{u}^{(a)} \odot \mathbf{u}^{(i)}; \mathbf{f}(a, i, \mathbf{w})]), \quad [15.32]$$

where $\mathbf{u}^{(a)}$ and $\mathbf{u}^{(i)}$ are the embeddings for spans a and i respectively, as defined in Equation 15.25.

- 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

8419 pairs, constructing a cluster-pair encoding by **pooling** over the mention-pair encodings
8420 for all pairs of mentions within the two clusters.

8421 **15.3.2.2 Entity embeddings**

8422 In entity-based coreference resolution, each entity should be represented by properties of
8423 its mentions. In a distributed setting, we maintain a set of vector entity embeddings, v_e .
8424 Each candidate mention receives an embedding u_i ; Wiseman et al. (2016) compute this
8425 embedding by a single-layer neural network, applied to a vector of surface features. The
8426 decision of whether to merge mention i with entity e can then be driven by a feedforward
8427 network, $\psi_E(i, e) = \text{Feedforward}([v_e; u_i])$. If i is added to entity e , then its representa-
8428 tion is updated recurrently, $v_e \leftarrow f(v_e, u_i)$, using a recurrent neural network such as a
8429 long short-term memory (LSTM; chapter 6). Alternatively, we can apply a **pooling** oper-
8430 ation, such as max-pooling or average-pooling (chapter 3), setting $v_e \leftarrow \text{Pool}(v_e, u_i)$. In
8431 either case, the update to the representation of entity e can be thought of as adding new
8432 information about the entity from mention i .

8433 **15.4 Evaluating coreference resolution**

8434 The state of coreference evaluation is aggravatingly complex. Early attempts at sim-
8435 ple evaluation metrics were found to be susceptible to trivial baselines, such as placing
8436 each mention in its own cluster, or grouping all mentions into a single cluster. Follow-
8437 ing Denis and Baldridge (2009), the CoNLL 2011 shared task on coreference (Pradhan
8438 et al., 2011) formalized the practice of averaging across three different metrics: MUC (Vi-
8439 lain et al., 1995), B-CUBED (Bagga and Baldwin, 1998a), and CEAf (Luo, 2005). Refer-
8440 ence implementations of these metrics are available from Pradhan et al. (2014) at <https://github.com/conll/reference-coreference-scorers>.
8441

8442 **Additional resources**

8443 Ng (2010) surveys coreference resolution through 2010. Early work focused exclusively
8444 on pronoun resolution, with rule-based (Lappin and Leass, 1994) and probabilistic meth-
8445 ods (Ge et al., 1998). The full coreference resolution problem was popularized in a shared
8446 task associated with the sixth Message Understanding Conference, which included coref-
8447 erence annotations for training and test sets of thirty documents each (Grishman and
8448 Sundheim, 1996). An influential early paper was the decision tree approach of Soon et al.
8449 (2001), who introduced mention ranking. A comprehensive list of surface features for
8450 coreference resolution is offered by Bengtson and Roth (2008). Durrett and Klein (2013)
8451 improved on prior work by introducing a large lexicalized feature set; subsequent work
8452 has emphasized neural representations of entities and mentions (Wiseman et al., 2015).

8453 **Exercises**

- 8454 1. Select an article from today's news, and annotate coreference for the first twenty
8455 noun phrases that appear in the article (include nested noun phrases). That is,
8456 group the noun phrases into entities, where each entity corresponds to a set of noun
8457 phrases. Then specify the mention-pair training data that would result from the first
8458 five noun phrases.
 - 8459 2. Using your annotations from the preceding problem, compute the following statistics:
8460
 - 8461 • The number of times new entities are introduced by each of the three types of
8462 referring expressions: pronouns, proper nouns, and nominals. Include "single-
8463 ton" entities that are mentioned only once.
 - 8464 • For each type of referring expression, compute the fraction of mentions that are
8465 anaphoric.
 - 8466 3. Apply a simple heuristic to all pronouns in the article from the previous exercise.
8467 Specifically, link each pronoun to the closest preceding noun phrase that agrees in
8468 gender, number, animacy, and person. Compute the following evaluation:
8469
 - 8470 • True positive: a pronoun that is linked to a noun phrase with which it is coref-
erent, or is correctly labeled as the first mention of an entity.
 - 8471 • False positive: a pronoun that is linked to a noun phrase with which it is not
8472 coreferent. (This includes mistakenly linking singleton or non-referential pro-
8473 nouns.)
 - 8474 • False negative: a pronoun that is not linked to a noun phrase with which it is
8475 coreferent.
- 8476 Compute the *F-MEASURE* for your method, and for a trivial baseline in which ev-
8477 ery mention is its own entity. Are there any additional heuristics that would have
8478 improved the performance of this method?
- 8479 4. Durrett and Klein (2013) compute the probability of the gold coreference clustering
8480 by summing over all antecedent structures that are compatible with the clustering.
8481 For example, if there are three mentions of a single entity, m_1, m_2, m_3 , there are two
8482 possible antecedent structures: $a_2 = 1, a_3 = 1$ and $a_2 = 1, a_3 = 2$. Compute the
8483 number of antecedent structures for a single entity with K mentions.
 - 8484 5. Suppose that all mentions can be unambiguously divided into C classes, for exam-
8485 ple by gender and number. Further suppose that mentions from different classes
8486 can never corefer. In a document with M mentions, give upper and lower bounds
8487 on the total number of possible coreference clusterings, in terms of the Bell numbers

and the parameters M and C . Compute numerical upper and lower bounds for the case $M = 4, C = 2$.

6. Lee et al. (2017) propose a model that considers all contiguous spans in a document as possible mentions.

- a) In a document of length M , how many mention pairs must be evaluated? (All answers can be given in asymptotic, big-O notation.)
- b) To make inference more efficient, Lee et al. (2017) restrict consideration to spans of maximum length $L \ll M$. Under this restriction, how many mention pairs must be evaluated?
- c) To further improve inference, one might evaluate coreference only between pairs of mentions whose endpoints are separated by a maximum of D tokens. Under this additional restriction, how many mention pairs must be evaluated?

7. In Spanish, the subject can be omitted when it is clear from context, e.g.,

(15.21) *Las ballenas no son peces. Son mamíferos.*

The whales no are fish. Are mammals.

Whales are not fish. They are mammals.

Resolution of such **null subjects** is facilitated by the Spanish system of verb morphology, which includes distinctive suffixes for most combinations of person and number. For example, the verb form *son* ('are') agrees with the third-person plural pronouns *ellos* (masculine) and *ellas* (feminine), as well as the second-person plural *ustedes*.

Suppose that you are given the following components:

- A system that automatically identifies verbs with null subjects.
- A function $c(j, p) \in \{0, 1\}$ that indicates whether pronoun p is compatible with null subject j , according to the verb morphology.
- A trained mention-pair model, which computes scores $\psi(w_i, w_j, i - j) \in \mathbb{R}$ for all pairs of mentions i and j , scoring the pair by the antecedent mention w_i , the anaphor w_j , and the distance $i - j$.

Describe an integer linear program that simultaneously performs two tasks: resolving coreference among all entity mentions, and identifying suitable pronouns for all null subjects. In the example above, your program should link the null subject with *las ballenas* ('whales'), and identify *ellas* as the correct pronoun. For simplicity, you may assume that null subjects cannot be antecedents, and you need not worry about the transitivity constraint described in § 15.2.3.

- 8521 8. Use the policy gradient algorithm to compute the gradient for the following sce-
 8522 nario, based on the Bell tree in Figure 15.4:

- 8523 • The gold clustering c^* is $\{Abigail, her\}, \{she\}$.
 8523 • Drawing a single sequence of actions ($K = 1$) from the current policy, you obtain the following incremental clusterings:

$$\begin{aligned} c(a_1) &= \{Abigail\} \\ c(\mathbf{a}_{1:2}) &= \{Abigail, she\} \\ c(\mathbf{a}_{1:3}) &= \{Abigail, she\}, \{her\}. \end{aligned}$$

- 8524 • At each mention t , the action space \mathcal{A}_t is to merge the mention with each exist-
 8525 ing cluster, or the empty cluster, with probability,

$$\Pr(\text{Merge}(m_t, c(\mathbf{a}_{1:t-1}))) \propto \exp \psi_E(m_t \cup c(\mathbf{a}_{1:t-1})), \quad [15.33]$$

8526 where the cluster score $\psi_E(m_t \cup c)$ is defined in Equation 15.15.

8527 Compute the gradient $\frac{\partial}{\partial \theta} L(\theta)$ in terms of the loss $\ell(c(\mathbf{a}))$ and the features of each
 8528 (potential) cluster. Explain the differences between the gradient-based update $\theta \leftarrow \theta - \frac{\partial}{\partial \theta} L(\theta)$
 8529 and the incremental perceptron update from this same example.

- 8530 9. As discussed in § 15.1.1.4, some pronouns are not referential. In English, this occurs
 8531 frequently with the word *it*. Download the text of *Alice in Wonderland* from nltk,
 8532 and examine the first ten appearances of *it*. For each occurrence:

- 8533 • First, examine a five-token window around the word. In the first example, this
 8534 window is,

8535 , but it had no

8536 Is there another pronoun that could be substituted for *it*? Consider *she*, *they*,
 8537 and *them*. In this case, both *she* and *they* yield grammatical substitutions. What
 8538 about the other ten cases?

- 8539 • Now, view an fifteen-word window for each example. Based on this window,
 8540 mark whether you think the word *it* is referential.

8541 How often does the substitution test predict whether *it* is referential?

- 8542 10. Now try to automate the test, using the Google n -grams corpus (BRANTS and Franz,
 8543 2006). Specifically, find the count of each 5-gram containing *it*, and then compute the
 8544 counts of 5-grams in which *it* is replaced with other third-person pronouns: *he*, *she*,
 8545 *they*, *her*, *him*, *them*, *herself*, *himself*. There are various ways to get these counts.

8546 One approach is to download the raw data and search it; another is to construct web
8547 queries to <https://books.google.com/ngrams>.

8548 Compare the ratio of the counts of the original 5-gram to the summed counts of
8549 the 5-grams created by substitution. Is this ratio a good predictor of whether *it* is
8550 referential?

8551 Chapter 16

8552 Discourse

8553 Applications of natural language processing often concern multi-sentence documents:
8554 from paragraph-long restaurant reviews, to 500-word newspaper articles, to 500-page
8555 novels. Yet most of the methods that we have discussed thus far are concerned with
8556 individual sentences. This chapter discusses theories and methods for handling multi-
8557 sentence linguistic phenomena, known collectively as **discourse**. There are diverse char-
8558 acterizations of discourse structure, and no single structure is ideal for every computa-
8559 tional application. This chapter covers some of the most well studied discourse repre-
8560 sentations, while highlighting computational models for identifying and exploiting these
8561 structures.

8562 16.1 Segments

8563 A document or conversation can be viewed as a sequence of **segments**, each of which is
8564 **cohesive** in its content and/or function. In Wikipedia biographies, these segments often
8565 pertain to various aspects to the subject's life: early years, major events, impact on others,
8566 and so on. This segmentation is organized around **topics**. Alternatively, scientific research
8567 articles are often organized by **functional themes**: the introduction, a survey of previous
8568 research, experimental setup, and results.

8569 Written texts often mark segments with section headers and related formatting de-
8570 vices. However, such formatting may be too coarse-grained to support applications such
8571 as the retrieval of specific passages of text that are relevant to a query (Hearst, 1997).
8572 Unformatted speech transcripts, such as meetings and lectures, are also an application
8573 scenario for segmentation (Carletta, 2007; Glass et al., 2007; Janin et al., 2003).

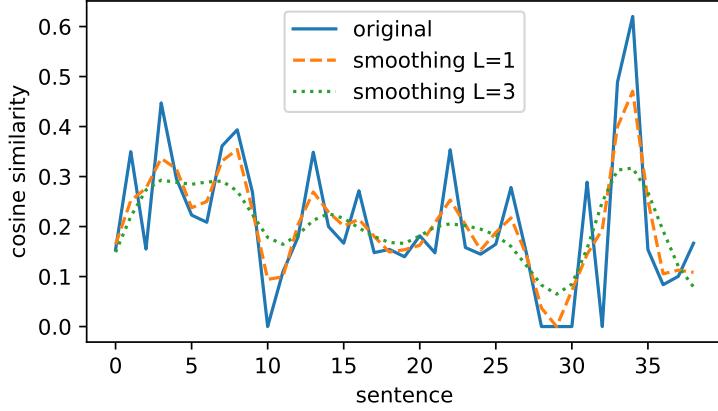


Figure 16.1: Smoothed cosine similarity among adjacent sentences in a news article. Local minima at $m = 10$ and $m = 29$ indicate likely segmentation points.

8574 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,

$$s_m = \frac{\mathbf{x}_m \cdot \mathbf{x}_{m+1}}{\|\mathbf{x}_m\|_2 \times \|\mathbf{x}_{m+1}\|_2} \quad [16.1]$$

$$\bar{s}_m = \sum_{\ell=0}^L k_\ell (s_{m+\ell} + s_{m-\ell}), \quad [16.2]$$

8575 with k_ℓ representing the value of a smoothing kernel of size L , e.g. $\mathbf{k} = [1, 0.5, 0.25]^\top$.
 8576 Segmentation points are then identified at local minima in the smoothed similarities \bar{s} ,
 8577 since these points indicate changes in the overall distribution of words in the text. An
 8578 example is shown in Figure 16.1.

8579 Text segmentation can also be formulated as a probabilistic model, in which each seg-
 8580 ment has a unique language model that defines the probability over the text in the seg-
 8581 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

8582 segmentation achieves high likelihood by grouping segments with similar word distribu-
8583 tions. This probabilistic approach can be extended to **hierarchical topic segmentation**, in
8584 which each topic segment is divided into subsegments (Eisenstein, 2009). All of these ap-
8585 proaches are unsupervised. While labeled data can be obtained from well-formatted texts
8586 such as textbooks, such annotations may not generalize to speech transcripts in alterna-
8587 tive domains. Supervised methods have been tried in cases where in-domain labeled data
8588 is available, substantially improving performance by learning weights on multiple types
8589 of features (Galley et al., 2003).

8590 16.1.2 Functional segmentation

8591 In some genres, there is a canonical set of communicative *functions*: for example, in sci-
8592 entific research articles, one such function is to communicate the general background for
8593 the article, another is to introduce a new contribution, or to describe the aim of the re-
8594 search (Teufel et al., 1999). A **functional segmentation** divides the document into con-
8595 tiguous segments, sometimes called **rhetorical zones**, in which each sentence has the same
8596 function. Teufel and Moens (2002) train a supervised classifier to identify the functional
8597 of each sentence in a set of scientific research articles, using features that describe the sen-
8598 tence's position in the text, its similarity to the rest of the article and title, tense and voice of
8599 the main verb, and the functional role of the previous sentence. Functional segmentation
8600 can also be performed without supervision. Noting that some types of Wikipedia arti-
8601 cles have very consistent functional segmentations (e.g., articles about cities or chemical
8602 elements), Chen et al. (2009) introduce an unsupervised model for functional segmenta-
8603 tion, which learns both the language model associated with each function and the typical
8604 patterning of functional segments across the article.

8605 16.2 Entities and reference

8606 Another dimension of discourse relates to which entities are mentioned throughout the
8607 text, and how. Consider the examples in Figure 16.2: Grosz et al. (1995) argue that the first
8608 discourse is more coherent. Do you agree? The examples differ in their choice of **refe-
8609 riring expressions** for the protagonist *John*, and in the syntactic constructions in sentences
8610 (b) and (d). The examples demonstrate the need for theoretical models to explain how
8611 referring expressions are chosen, and where they are placed within sentences. Such mod-
8612 els can then be used to help interpret the overall structure of the discourse, to measure
8613 discourse coherence, and to generate discourses in which referring expressions are used
8614 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.

8615 16.2.1 Centering theory

8616 The relationship between discourse and entity reference is most elaborated in **centering**
 8617 **theory** (Grosz et al., 1995). According to the theory, every utterance in the discourse is
 8618 characterized by a set of entities, known as *centers*.

- 8619 • The **forward-looking centers** in utterance m are all the entities that are mentioned
 8620 in the utterance, $c_f(w_m) = \{e_1, e_2, \dots\}$. The forward-looking centers are partially
 8621 ordered by their syntactic prominence, favoring subjects over other positions.
 8622 • The **backward-looking center** $c_b(w_m)$ is the highest-ranked element in the set of
 8623 forward-looking centers from the previous utterance $c_f(w_{m-1})$ that is also men-
 8624 tioned in w_m .

8625 Given these two definitions, centering theory makes the following predictions about
 8626 the form and position of referring expressions:

- 8627 1. If a pronoun appears in the utterance w_m , then the backward-looking center $c_b(w_m)$
 8628 must also be realized as a pronoun. This rule argues against the use of *it* to refer
 8629 to the piano store in Example (16.2d), since JOHN is the backward looking center of
 8630 (16.2d), and he is mentioned by name and not by a pronoun.
 8631 2. Sequences of utterances should retain the same backward-looking center if possible,
 8632 and ideally, the backward-looking center should also be the top-ranked element in
 8633 the list of forward-looking centers. This rule argues in favor of the preservation of
 8634 JOHN as the backward-looking center throughout Example (16.1).

8635 Centering theory unifies aspects of syntax, discourse, and anaphora resolution. However,
 8636 it can be difficult to clarify exactly how to rank the elements of each utterance, or even
 8637 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

8656 text (Lapata, 2003), and disentangling multiple conversational threads in an online multi-
 8657 party chat (Elsner and Charniak, 2010).

8658 **16.2.3 *Formal semantics beyond the sentence level**

8659 An alternative view of the role of entities in discourse focuses on formal semantics, and the
 8660 construction of meaning representations for multi-sentence units. Consider the following
 8661 two sentences (from Bird et al., 2009):

- 8662 (16.3) a. Angus owns a dog.
 8663 b. It bit Irene.

8664 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:

$$\exists x. \text{DOG}(x) \wedge \text{OWN}(\text{ANGUS}, x) \quad [16.5]$$

$$\text{BITE}(y, \text{IRENE}). \quad [16.6]$$

8665 Unifying these two representations into the form of Equation 16.4 requires linking the
 8666 unbound variable y from [16.6] with the quantified variable x in [16.5]. Discourse under-
 8667 standing therefore requires the reader to update a set of assignments, from variables
 8668 to entities. This update would (presumably) link the *dog* in the first sentence of [16.3]
 8669 with the unbound variable y in the second sentence, thereby licensing the conjunction in
 8670 [16.4].² This basic idea is at the root of **dynamic semantics** (Groenendijk and Stokhof,
 8671 1991). **Segmented discourse representation theory** links dynamic semantics with a set
 8672 of **discourse relations**, which explain how adjacent units of text are rhetorically or con-
 8673 ceptually related (Lascarides and Asher, 2007). The next section explores the theory of
 8674 discourse relations in more detail.

8675 **16.3 Relations**

8676 In dependency grammar, sentences are characterized by a graph (usually a tree) of syntac-
 8677 tic relations between words, such as NSUBJ and DET. A similar idea can be applied at the
 8678 document level, identifying relations between discourse units, such as clauses, sentences,
 8679 or paragraphs. The task of **discourse parsing** involves identifying discourse units and
 8680 the relations that hold between them. These relations can then be applied to tasks such as
 8681 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.

8682 16.3.1 Shallow discourse relations

8683 The existence of discourse relations is hinted by **discourse connectives**, such as *however*,
 8684 *moreover*, *meanwhile*, and *if . . . then*. These connectives explicitly specify the relationship
 8685 between adjacent units of text: *however* signals a contrastive relationship, *moreover* signals
 8686 that the subsequent text elaborates or strengthens the point that was made immediately
 8687 beforehand, *meanwhile* indicates that two events are contemporaneous, and *if . . . then* sets
 8688 up a conditional relationship. Discourse connectives can therefore be viewed as a starting
 8689 point for the analysis of discourse relations.

8690 In **lexicalized tree-adjoining grammar for discourse (D-LTAG)**, each connective an-
 8691 chors a relationship between two units of text (Webber, 2004). This model provides the
 8692 theoretical basis for the **Penn Discourse Treebank (PDTB)**, the largest corpus of discourse
 8693 relations in English (Prasad et al., 2008). It includes a hierarchical inventory of discourse
 8694 relations (shown in Table 16.1), which is created by abstracting the meanings implied by
 8695 the discourse connectives that appear in real texts (Knott, 1996). These relations are then
 8696 annotated on the same corpus of news text used in the Penn Treebank (see § 9.2.2), adding
 8697 the following information:

- Each connective is annotated for the discourse relation or relations that it expresses, if any — many discourse connectives have senses in which they do not signal a discourse relation (Pitler and Nenkova, 2009).
- For each discourse relation, the two arguments of the relation are specified as ARG1 and ARG2, where ARG2 is constrained to be adjacent to the connective. These arguments may be sentences, but they may also smaller or larger units of text.
- Adjacent sentences are annotated for **implicit discourse relations**, which are not marked by any connective. When a connective could be inserted between a pair of sentence, the annotator supplies it, and also labels its sense (e.g., example 16.5). In some cases, there is no relationship at all between a pair of adjacent sentences; in other cases, the only relation is that the adjacent sentences mention one or more shared entity. These phenomena are annotated as NOREL and ENTRREL (entity relation), respectively.

Examples of Penn Discourse Treebank annotations are shown in (16.4). In (16.4), the word *therefore* acts as an explicit discourse connective, linking the two adjacent units of text. The Treebank annotations also specify the “sense” of each relation, linking the connective to a relation in the sense inventory shown in Table 16.1: in (16.4), the relation is PRAGMATIC CAUSE:JUSTIFICATION because it relates to the author’s communicative intentions. The word *therefore* can also signal causes in the external world (e.g., *He was therefore forced to relinquish his plan*). In **discourse sense classification**, the goal is to determine which discourse relation, if any, is expressed by each connective. A related task is the classification of implicit discourse relations, as in (16.5). In this example, the relationship between the adjacent sentences could be expressed by the connective *because*, indicating a CAUSE:REASON relationship.

16.3.1.1 Classifying explicit discourse relations and their arguments

As suggested by the examples above, many connectives can be used to invoke multiple types of discourse relations. Similarly, some connectives have senses that are unrelated to discourse: for example, *and* functions as a discourse connective when it links propositions, but not when it links noun phrases (Lin et al., 2014). Nonetheless, the senses of explicitly-marked discourse relations in the Penn Treebank are relatively easy to classify, at least at the coarse-grained level. When classifying the four top-level PDTB relations, 90% accuracy can be obtained simply by selecting the most common relation for each connective (Pitler and Nenkova, 2009). At the more fine-grained levels of the discourse relation hierarchy, connectives are more ambiguous. This fact is reflected both in the accuracy of automatic sense classification (Versley, 2011) and in interannotator agreement, 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).

8734 A more challenging task for explicitly-marked discourse relations is to identify the
 8735 scope of the arguments. Discourse connectives need not be adjacent to ARG1, as shown
 8736 in item 16.6, where ARG1 follows ARG2; furthermore, the arguments need not be contigu-
 8737 ous, as shown in (16.7). For these reasons, recovering the arguments of each discourse
 8738 connective is a challenging subtask. Because intra-sentential arguments are often syn-
 8739 tactic constituents (see chapter 10), many approaches train a classifier to predict whether
 8740 each constituent is an appropriate argument for each explicit discourse connective (Well-
 8741 ner and Pustejovsky, 2007; Lin et al., 2014, e.g.,).

8742 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).

8743 This basic framework can be instantiated in several ways, including both feature-based
 8744 and neural encoders. Several recent approaches are compared in the 2015 and 2016 shared
 8745 tasks at the Conference on Natural Language Learning (Xue et al., 2015, 2016).

8746 **Feature-based approaches** Each argument can be encoded into a vector of surface fea-
 8747 tures. The encoding typically includes lexical features (all words, or all content words, or
 8748 a subset of words such as the first three and the main verb), Brown clusters of individ-
 8749 ual words (§ 14.4), and syntactic features such as terminal productions and dependency
 8750 arcs (Pitler et al., 2009; Lin et al., 2009; Rutherford and Xue, 2014). The classification func-
 8751 tion then has two parts. First, it creates a joint feature vector by combining the encodings
 8752 of each argument, typically by computing the cross-product of all features in each encod-
 8753 ing:

$$\mathbf{f}(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}) = \{(a \times b \times y) : (\mathbf{z}_a^{(i)} \mathbf{z}_b^{(i+1)})\} \quad [16.10]$$

8754 The size of this feature set grows with the square of the size of the vocabulary, so it can be
 8755 helpful to select a subset of features that are especially useful on the training data (Park
 8756 and Cardie, 2012). After \mathbf{f} is computed, any classifier can be trained to compute the final
 8757 score, $\Psi(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}) = \boldsymbol{\theta} \cdot \mathbf{f}(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)})$.

8758 **Neural network approaches** In neural network architectures, the encoder is learned
 8759 jointly with the classifier as an end-to-end model. Each argument can be encoded using
 8760 a variety of neural architectures (surveyed in § 14.8): recursive (§ 10.6.1; Ji and Eisenstein,
 8761 2015), recurrent (§ 6.3; Ji et al., 2016), and convolutional (§ 3.4; Qin et al., 2017). The clas-
 8762 sification function can then be implemented as a feedforward neural network on the two
 8763 encodings (chapter 3; for examples, see Rutherford et al., 2017; Qin et al., 2017), or as a
 8764 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
 8765 encoding model can be trained by backpropagation from the classification objective, such
 8766 as the margin loss. Rutherford et al. (2017) show that neural architectures outperform
 8767 feature-based approaches in most settings. While neural approaches require engineering
 8768 the network architecture (e.g., embedding size, number of hidden units in the classifier),
 8769 feature-based approaches also require significant engineering to incorporate linguistic re-
 8770 sources such as Brown clusters and parse trees, and to select a subset of relevant features.

8771 16.3.2 Hierarchical discourse relations

8772 In sentence parsing, adjacent phrases combine into larger constituents, ultimately pro-
 8773 ducing a single constituent for the entire sentence. The resulting tree structure enables
 8774 structured analysis of the sentence, with subtrees that represent syntactically coherent
 8775 chunks of meaning. **Rhetorical Structure Theory (RST)** extends this style of hierarchical
 8776 analysis to the discourse level (Mann and Thompson, 1988).

8777 The basic element of RST is the **discourse unit**, which refers to a contiguous span of
 8778 text. **Elementary discourse units** (EDUs) are the atomic elements in this framework, and
 8779 are typically (but not always) clauses.⁴ Each discourse relation combines two or more
 8780 adjacent discourse units into a larger, composite discourse unit; this process ultimately
 8781 unites the entire text into a tree-like structure.⁵

8782 **Nuclearity** In many discourse relations, one argument is primary. For example:

8783 (16.8) [LaShawn loves animals]_N
 8784 [She has nine dogs and one pig]_S

8785 In this example, the second sentence provides EVIDENCE for the point made in the first
 8786 sentence. The first sentence is thus the **nucleus** of the discourse relation, and the second
 8787 sentence is the **satellite**. The notion of **nuclearity** is analogous to the head-modifier struc-
 8788 ture of dependency parsing (see § 11.1.1). However, in RST, some relations have multiple
 8789 nuclei. For example, the arguments of the CONTRAST relation are equally important:

8790 (16.9) [The clash of ideologies survives this treatment]_N
 8791 [but the nuance and richness of Gorky's individual characters have vanished in the scuffle]_N⁶

8792 Relations that have multiple nuclei are called **coordinating**; relations with a single nu-
 8793 cleus are called **subordinating**. Subordinating relations are constrained to have only two
 8794 arguments, while coordinating relations (such as CONJUNCTION) may have more than
 8795 two.

8796 **RST Relations** Rhetorical structure theory features a large inventory of discourse rela-
 8797 tions, which are divided into two high-level groups: subject matter relations, and presen-
 8798 tational relations. Presentational relations are organized around the intended beliefs of
 8799 the reader. For example, in (16.8), the second discourse unit provides evidence intended
 8800 to increase the reader's belief in the proposition expressed by the first discourse unit, that
 8801 *LaShawn loves animals*. In contrast, subject-matter relations are meant to communicate ad-
 8802 ditional facts about the propositions contained in the discourse units that they relate:

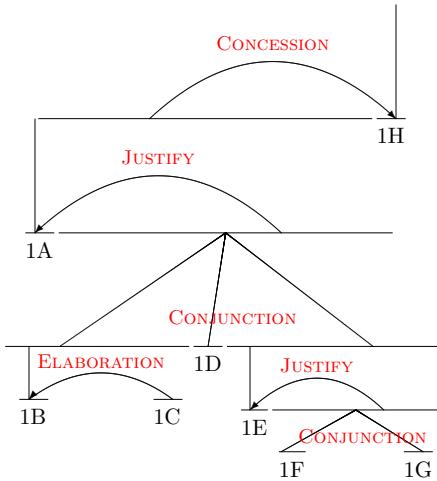
⁴Details of discourse segmentation can be found in the RST annotation manual (Carlson and Marcu, 2001).

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.

8803 (16.10) [the debt plan was rushed to completion]_N
 8804 [in order to be announced at the meeting]_S⁷

8805 In this example, the satellite describes a world state that is realized by the action described
 8806 in the nucleus. This relationship is about the world, and not about the author’s commu-
 8807 nicative intentions.

8808 **Example** Figure 16.5 depicts an RST analysis of a paragraph from a movie review. Asym-
 8809 metric (subordinating) relations are depicted with an arrow from the satellite to the nu-
 8810 cleus; symmetric (coordinating) relations are depicted with lines. The elementary dis-
 8811 course units 1F and 1G are combined into a larger discourse unit with the symmetric
 8812 CONJUNCTION relation. The resulting discourse unit is then the satellite in a JUSTIFY
 8813 relation with 1E.

⁷from the RST Treebank (Carlson et al., 2002)

8814 **16.3.2.1 Hierarchical discourse parsing**

8815 The goal of discourse parsing is to recover a hierarchical structural analysis from a doc-
 8816 ument text, such as the analysis in Figure 16.5. For now, let's assume a segmentation
 8817 of the document into elementary discourse units (EDUs); segmentation algorithms are
 8818 discussed below. After segmentation, discourse parsing can be viewed as a combination
 8819 of two components: the discourse relation classification techniques discussed in § 16.3.1.2,
 8820 and algorithms for phrase-structure parsing, such as chart parsing and shift-reduce, which
 8821 were discussed in chapter 10.

8822 Both chart parsing and shift-reduce require encoding composite discourse units, ei-
 8823 ther in a discrete feature vector or a dense neural representation.⁸ Some discourse parsers
 8824 rely on the **strong compositionality criterion** (Marcu, 1996), which states the assumption
 8825 that a composite discourse unit can be represented by its nucleus. This criterion is used in
 8826 feature-based discourse parsing to determine the feature vector for a composite discourse
 8827 unit (Hernault et al., 2010); it is used in neural approaches to setting the vector encod-
 8828 ing for a composite discourse unit equal to the encoding of its nucleus (Ji and Eisenstein,
 8829 2014). An alternative neural approach is to learn a composition function over the compo-
 8830 nents of a composite discourse unit (Li et al., 2014), using a recursive neural network (see
 8831 § 14.8.3).

8832 **Bottom-up discourse parsing** Assume a segmentation of the text into N elementary
 8833 discourse units with base representations $\{z^{(i)}\}_{i=1}^N$, and assume a composition function
 8834 COMPOSE $(z^{(i)}, z^{(j)}, \ell)$, which maps two encodings and a discourse relation ℓ into a new
 8835 encoding. The composition function can follow the strong compositionality criterion and
 8836 simply select the encoding of the nucleus, or it can do something more complex. We
 8837 also need a scoring function $\Psi(z^{(i,k)}, z^{(k,j)}, \ell)$, which computes a scalar score for the (bi-
 8838 narized) discourse relation ℓ with left child covering the span $i + 1 : k$, and the right
 8839 child covering the span $k + 1 : j$. Given these components, we can construct vector rep-
 8840 resentations for each span, and this is the basic idea underlying **compositional vector**
 8841 **grammars** (Socher et al., 2013).

8842 These same components can also be used in bottom-up parsing, in a manner that is
 8843 similar to the CKY algorithm for weighted context-free grammars (see § 10.1): compute
 8844 the score and best analysis for each possible span of increasing lengths, while storing
 8845 back-pointers that make it possible to recover the optimal parse of the entire input. How-
 8846 ever, there is an important distinction from CKY parsing: for each labeled span (i, j, ℓ) , we
 8847 must use the composition function to construct a representation $z^{(i,j,\ell)}$. This representa-
 8848 tion is then used to combine the discourse unit spanning $i + 1 : j$ in higher-level discourse
 8849 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).

8850 ning $i + 1 : j$, and this violates the locality assumption that underlie CKY’s optimality
 8851 guarantee. Bottom-up parsing with recursively constructed span representations is gen-
 8852 erally not guaranteed to find the best-scoring discourse parse. This problem is explored
 8853 in an exercise at the end of the chapter.

8854 **Transition-based discourse parsing** One drawback of bottom-up parsing is its cubic
 8855 time complexity in the length of the input. For long documents, transition-based parsing
 8856 is an appealing alternative. The shift-reduce algorithm can be applied to discourse parsing
 8857 fairly directly (Sagae, 2009): the stack stores a set of discourse units and their repres-
 8858 entations, and each action is chosen by a function of these representations. This function
 8859 could be a linear product of weights and features, or it could be a neural network ap-
 8860 plied to encodings of the discourse units. The REDUCE action then performs composition
 8861 on the two discourse units at the top of the stack, yielding a larger composite discourse
 8862 unit, which goes on top of the stack. All of the techniques for integrating learning and
 8863 transition-based parsing, described in § 11.3, are applicable to discourse parsing.

8864 16.3.2.2 Segmenting discourse units

8865 In rhetorical structure theory, elementary discourse units do not cross the sentence bound-
 8866 ary, so discourse segmentation can be performed within sentences, assuming the sentence
 8867 segmentation is given. The segmentation of sentences into elementary discourse units is
 8868 typically performed using features of the syntactic analysis (Braud et al., 2017). One ap-
 8869 proach is to train a classifier to determine whether each syntactic constituent is an EDU,
 8870 using features such as the production, tree structure, and head words (Soricut and Marcu,
 8871 2003; Hernault et al., 2010). Another approach is to train a sequence labeling model, such
 8872 as a conditional random field (Sporleder and Lapata, 2005; Xuan Bach et al., 2012; Feng
 8873 et al., 2014). This is done using the BIO formalism for segmentation by sequence labeling,
 8874 described in § 8.3.

8875 16.3.3 Argumentation

8876 An alternative view of text-level relational structure focuses on **argumentation** (Stab and
 8877 Gurevych, 2014b). Each segment (typically a sentence or clause) may support or rebut
 8878 another segment, creating a graph structure over the text. In the following example (from
 8879 Peldszus and Stede, 2013), segment S_2 provides argumentative support for the proposi-
 8880 tion in the segment S_1 :

8881 (16.11) [We should tear the building down] $_{S1}$
 8882 [because it is full of asbestos] $_{S2}$.

8883 Assertions may also support or rebut proposed links between two other assertions, cre-
 8884 ating a **hypergraph**, which is a generalization of a graph to the case in which edges can

8885 join any number of vertices. This can be seen by introducing another sentence into the
8886 example:

8887 (16.12) [In principle it is possible to clean it up.]_{S3}
8888 [but according to the mayor that is too expensive.]_{S4}

8889 S3 acknowledges the validity of *S2*, but **undercuts** its support of *S1*. This can be repre-
8890 sented by introducing a hyperedge, $(S3, S2, S1)_{\text{undercut}}$, indicating that *S3* undercuts the
8891 proposed relationship between *S2* and *S1*. *S4* then undercuts the relevance of *S3*.

8892 **Argumentation mining** is the task of recovering such structures from raw texts. At
8893 present, annotations of argumentation structure are relatively small: Stab and Gurevych
8894 (2014a) have annotated a collection of 90 persuasive essays, and Peldszus and Stede (2015)
8895 have solicited and annotated a set of 112 paragraph-length “microtexts” in German.

8896 16.3.4 Applications of discourse relations

8897 The predominant application of discourse parsing is to select content within a document.
8898 In rhetorical structure theory, the nucleus is considered the more important element of
8899 the relation, and is more likely to be part of a summary of the document; it may also
8900 be more informative for document classification. The D-LTAG theory that underlies the
8901 Penn Discourse Treebank lacks this notion of nuclearity, but arguments may have varying
8902 importance, depending on the relation type. For example, the span of text constituting
8903 ARG1 of an expansion relation is more likely to appear in a summary, while the sentence
8904 constituting ARG2 of an implicit relation is less likely (Louis et al., 2010). Discourse rela-
8905 tions may also signal segmentation points in the document structure. Explicit discourse
8906 markers have been shown to correlate with changes in subjectivity, and identifying such
8907 change points can improve document-level sentiment classification, by helping the clas-
8908 sifier to focus on the subjective parts of the text (Trivedi and Eisenstein, 2013; Yang and
8909 Cardie, 2014).

8910 16.3.4.1 Extractive Summarization

8911 Text **summarization** is the problem of converting a longer text into a shorter one, while
8912 still conveying the key facts, events, ideas, and sentiments from the original. In **extractive**
8913 **summarization**, the summary is a subset of the original text; in **abstractive summariza-**
8914 **tion**, the summary is produced *de novo*, by paraphrasing the original, or by first encoding
8915 it into a semantic representation (see § 19.2). The main strategy for extractive summa-
8916 rization is to maximize **coverage**, choosing a subset of the document that best covers the
8917 concepts mentioned in the document as a whole; typically, coverage is approximated by
8918 bag-of-words overlap (Nenkova and McKeown, 2012). Coverage-based objectives can be
8919 supplemented by hierarchical discourse relations, using the principle of nuclearity: in any
8920 subordinating discourse relation, the nucleus is more critical to the overall meaning of the

8921 text, and is therefore more important to include in an extractive summary (Marcu, 1997a).⁹
 8922 This insight can be generalized from individual relations using the concept of **discourse**
 8923 **depth** (Hirao et al., 2013): for each elementary discourse unit e , the discourse depth d_e is
 8924 the number of relations in which a discourse unit containing e is the satellite.

8925 Both discourse depth and nuclearity can be incorporated into extractive summarization
 8926 using constrained optimization. Let \mathbf{x}_n be a bag-of-words vector representation of
 8927 elementary discourse unit n , let $y_n \in \{0, 1\}$ indicate whether n is included in the summary,
 8928 and let d_n be the depth of unit n . Furthermore, let each discourse unit have a “head” h ,
 8929 which is defined recursively:

- 8930 • if a discourse unit is produced by a subordinating relation, then its head is the head
 8931 of the (unique) nucleus;
- 8932 • if a discourse unit is produced by a coordinating relation, then its head is the head
 8933 of the left-most nucleus;
- 8934 • for each elementary discourse unit, its parent $\pi(n) \in \{\emptyset, 1, 2, \dots, N\}$ is the head of
 8935 the smallest discourse unit containing n whose head is not n ;
- 8936 • 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),

$$\begin{aligned} & \max_{\mathbf{y}=\{0,1\}^N} \sum_{n=1}^N y_n \frac{\Psi(\mathbf{x}_n, \{\mathbf{x}_{1:N}\})}{d_n} \\ & \text{s.t. } \sum_{n=1}^N y_n \left(\sum_{j=1}^V x_{n,j} \right) \leq L \\ & \quad y_{\pi(n)} \geq y_n, \quad \forall n \text{ s.t. } \pi(n) \neq \emptyset \end{aligned} \tag{16.11}$$

8937 where $\Psi(\mathbf{x}_n, \{\mathbf{x}_{1:N}\})$ measures the coverage of elementary discourse unit n with respect
 8938 to the rest of the document, and $\sum_{j=1}^V x_{n,j}$ is the number of tokens in \mathbf{x}_n . The first con-
 8939 straint ensures that the number of tokens in the summary has an upper bound L . The
 8940 second constraint ensures that no elementary discourse unit is included unless its parent
 8941 is also included. In this way, the discourse structure is used twice: to downweight the
 8942 contributions of elementary discourse units that are not central to the discourse, and to
 8943 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).

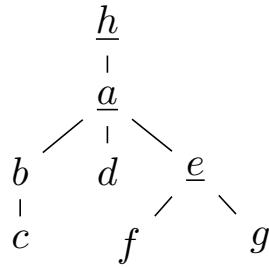


Figure 16.6: A **discourse depth tree** (Hirao et al., 2013) for the discourse parse from Figure 16.5, in which each elementary discourse unit is connected to its parent. The discourse units in one valid summary are underlined.

8944 mization problem in 16.11 can be solved with **integer linear programming**, described in
 8945 § 13.2.2.¹⁰

8946 Figure 16.6 shows a **discourse depth tree** for the RST analysis from Figure 16.5, in
 8947 which each elementary discourse is connected to (and below) its parent. The underlined
 8948 discourse units in the figure constitute the following summary:

8949 (16.13) It could have been a great movie, and I really liked the son of the leader of the
 8950 Samurai. But, other than all that, this movie is nothing more than hidden rip-offs.

8951 16.3.4.2 Document classification

8952 Hierarchical discourse structures lend themselves naturally to text classification: in a sub-
 8953 ordinating discourse relation, the nucleus should play a stronger role in the classification
 8954 decision than the satellite. Various implementations of this idea have been proposed.

- 8955 • Focusing on within-sentence discourse relations and lexicon-based classification (see
 8956 § 4.1.2), Voll and Taboada (2007) simply ignore the text in the satellites of each dis-
 8957 course relation.
- 8958 • At the document level, elements of each discourse relation argument can be reweighted,
 8959 favoring words in the nucleus, and disfavoring words in the satellite (Heerschop
 8960 et al., 2011; Bhatia et al., 2015). This approach can be applied recursively, computing
 8961 weights across the entire document. The weights can be relation-specific, so that the
 8962 features from the satellites of contrastive relations are discounted or even reversed.
- 8963 • Alternatively, the hierarchical discourse structure can define the structure of a **re-
 8964 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).

8965 discourse unit is computed from its arguments and from a parameter corresponding
 8966 to the discourse relation (Ji and Smith, 2017).

8967 Shallow, non-hierarchical discourse relations have also been applied to document clas-
 8968 sification. One approach is to impose a set of constraints on the analyses of individual
 8969 discourse units, so that adjacent units have the same polarity when they are connected
 8970 by a discourse relation indicating agreement, and opposite polarity when connected by a
 8971 contrastive discourse relation, indicating disagreement (Somasundaran et al., 2009; Zirn
 8972 et al., 2011). Yang and Cardie (2014) apply explicitly-marked relations from the Penn
 8973 Discourse Treebank to the problem of sentence-level sentiment polarity classification (see
 8974 § 4.1). They impose the following soft constraints:

- 8975 • When a CONTRAST relation appears at the beginning of a sentence, the sentence
 8976 should have the opposite sentiment polarity as its predecessor.
- 8977 • When an EXPANSION or CONTINGENCY appears at the beginning of a sentence, it
 8978 should have the same polarity as its predecessor.
- 8979 • When a CONTRAST relation appears *within* a sentence, the sentence should have
 8980 neutral polarity, since it is likely to express both sentiments.

8981 These discourse-driven constraints are shown to improve performance on two datasets of
 8982 product reviews.

8983 16.3.4.3 Coherence

8984 Just as **grammaticality** is the property shared by well-structured sentences, **coherence** is
 8985 the property shared by well-structured discourses. One application of discourse process-
 8986 ing is to measure (and maximize) the coherence of computer-generated texts like transla-
 8987 tions and summaries (Kibble and Power, 2004). Coherence assessment is also used to eval-
 8988 uate human-generated texts, such as student essays (e.g., Miltsakaki and Kukich, 2004;
 8989 Burstein et al., 2013).

8990 Coherence subsumes a range of phenomena, many of which have been highlighted
 8991 earlier in this chapter: e.g., that adjacent sentences should be lexically cohesive (Foltz
 8992 et al., 1998; Ji et al., 2015; Li and Jurafsky, 2017), and that entity references should follow
 8993 the principles of centering theory (Barzilay and Lapata, 2008; Nguyen and Joty, 2017).
 8994 Discourse relations also bear on the coherence of a text in a variety of ways:

- 8995 • Hierarchical discourse relations tend to have a “canonical ordering” of the nucleus
 8996 and satellite (Mann and Thompson, 1988): for example, in the ELABORATION rela-
 8997 tion from rhetorical structure theory, the nucleus always comes first, while in the
 8998 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/>

9030 Exercises

- 9031 1. One way to formulate text segmentation as a probabilistic model is through the use
 9032 of the Dirichlet compound Multinomial (DCM) distribution, which computes the probability
 9033 of a bag-of-words, $\text{DCM}(\mathbf{x}; \boldsymbol{\alpha})$, where $\boldsymbol{\alpha}$ is a parameter, $\boldsymbol{\alpha} \in \mathbb{R}_+^V$. This
 9034 distribution can be configured to assign high likelihood to bag-of-words vectors that
 9035 are internally coherent, such that individual words appear repeatedly: for example,
 9036 this behavior can be observed for simple parameterizations, such as $\boldsymbol{\alpha} = \{\alpha_i\}_{i=1}^V$.

9037 Let $\psi_\alpha(i, j)$ represent the log-probability of a segment $w_{i+1:j}$ under a DCM distribution
 9038 with parameter $\boldsymbol{\alpha}$. Give a dynamic program for segmenting a text into a total
 9039 of K segments maximizing the sum of log-probabilities $\sum_{k=1}^K \psi_\alpha(s_{k-1}, s_k)$, where
 9040 s_k indexes the last token of segment k , and $s_0 = 0$. The time complexity of your
 9041 dynamic program should not be worse than quadratic in the length of the input and
 9042 linear in the number of segments.

- 9043 2. Building on the previous problem, you will now adapt the CKY algorithm to per-
 9044 form hierarchical segmentation. Define a hierarchical segmentation as a set of seg-
 9045 mentations $\{\{s_k^{(\ell)}\}_{k=1}^{K^{(\ell)}}\}_{\ell=1}^L$, where L is the segmentation depth. To ensure that the
 9046 segmentation is hierarchically valid, we require that each segmentation point $s_k^{(\ell)}$
 9047 at level ℓ is also a segmentation point at level $\ell - 1$, where $\ell > 1$. We will focus
 9048 on binary hierarchical segmentation, so that each segment at level $\ell > 1$ has ex-
 9049 actly 2 subsegments. Define the score of a hierarchical segmentation as the sum of
 9050 the scores of all segments (at all levels), using the the DCM log-probabilities from
 9051 the previous problem as the segment scores. Give a CKY-like recurrence such that
 9052 the optimal “parse” of the text is the maximum log-probability binary segmentation
 9053 with exactly L levels.
- 9054 3. In this exercise, you will produce a figure similar to Figure 16.1.
- 9055 a) Implement the smoothed cosine similarity metric from Equation 16.2, using the
 9056 smoothing kernel $\mathbf{k} = [.5, .3, .15, .05]$.
- 9057 b) Download the text of a news article with at least ten paragraphs.
- 9058 c) Compute and plot the smoothed similarity \bar{s} over the length of the article.
- 9059 d) Identify *local minima* in \bar{s} as follows: first find all sentences m such that $\bar{s}_m <$
 9060 $\bar{s}_{m \pm 1}$. Then search among these points to find the five sentences with the lowest
 9061 \bar{s}_m .
- 9062 e) How often do the five local minima correspond to paragraph boundaries?
 9063 • The fraction of local minima that are paragraph boundaries is the **precision-**
 9064 **at- k** , where in this case, $k = 5$.

- 9065 • The fraction of paragraph boundaries which are local minima is the **recall-**
9066 **at- k .**
9067 • Compute precision-at- k and recall-at- k for $k = 3$ and $k = 10$.

- 9068 4. This exercise is to be done in pairs. Each participant selects an article from to-
9069 day's news, and replaces all mentions of individual people with special tokens like
9070 PERSON1, PERSON2, and so on. The other participant should then use the rules
9071 of centering theory to guess each type of referring expression: full name (*Captain*
9072 *Ahab*), partial name (e.g., *Ahab*), nominal (e.g., *the ship's captain*), or pronoun. Check
9073 whether the predictions match the original text, and whether the text conforms to
9074 the rules of centering theory.
- 9075 5. The entity grid representation of centering theory can be used to compute a score for
9076 adjacent sentences, as described in § 16.2.2. Given a set of sentences, these scores can
9077 be used to compute an optimal ordering. Show that finding the ordering with the
9078 maximum log probability is NP-complete, by reduction from a well-known prob-
9079 lem.
- 9080 6. Some discourse connectives tend to occur between their arguments; others can pre-
9081 cede both arguments, and a few can follow both arguments. Indicate whether the
9082 following connectives can occur between, before, and after their arguments: *how-
9083 ever, but, while* (contrastive, not temporal), *although, therefore, nonetheless*.
- 9084 7. In § 16.3.2.1, it is noted that bottom-up parsing with compositional vector represen-
9085 tations of each span is not guaranteed to be optimal. In this exercise, you will con-
9086 struct a minimal example proving this point. Consider a discourse with four units,
9087 with base representations $\{z^{(i)}\}_{i=1}^4$. Construct a scenario in which the parse selected
9088 by bottom-up parsing is not optimal, and give the precise mathematical conditions
9089 under which this suboptimal parse is selected. You may ignore the relation labels ℓ
9090 for the purpose of this example.
- 9091 8. As noted in § 16.3.3, arguments can described by hypergraphs, in which a segment
9092 may **undercut** a proposed edge between two other segments. Extend the model of
9093 extractive summarization described in § 16.3.4.1 to arguments, adding the follwoing
9094 constraint: if segment i undercuts an argumentative relationship between j and k ,
9095 then i cannot be included in the summary unless both j and k are included. Your sol-
9096 uction should take the form of a set of *linear* constraints on an integer linear program
9097 — that is, each constraint can only involve addition and subtraction of variables.

9098 In the next two exercises, you will explore the use of discourse connectives in a real corpus.
9099 Using `nltk`, acquire the Brown corpus, and identify sentences that begin with any of the
9100 following connectives: *however, nevertheless, moreover, furthermore, thus*.

9101 9. Both lexical consistency and discourse connectives contribute to the **cohesion** of a
9102 text. We might therefore expect adjacent sentences that are joined by explicit dis-
9103 course connectives to also have higher word overlap. Using the Brown corpus, test
9104 this theory by computing the average cosine similarity between adjacent sentences
9105 that are connected by one of the connectives mentioned above. Compare this to the
9106 average cosine similarity of all other adjacent sentences. If you know how, perform
9107 a two-sample t-test to determine whether the observed difference is statistically sig-
9108 nificant.

9109 10. Group the above connectives into the following three discourse relations:

- 9110 • Expansion: *moreover, furthermore*
9111 • Comparison: *however, nevertheless*
9112 • Contingency: *thus*

9113 Focusing on pairs of sentences which are joined by one of these five connectives,
9114 build a classifier to predict the discourse relation from the text of the two adjacent
9115 sentences — taking care to ignore the connective itself. Use the first 30000 sentences
9116 of the Brown corpus as the training set, and the remaining sentences as the test
9117 set. Compare the performance of your classifier against simply choosing the most
9118 common class. Using a bag-of-words classifier, it is hard to do much better than this
9119 baseline, so consider alternatives!

9120

Part IV

9121

Applications

9122 Chapter 17

9123 Information extraction

9124 Computers offer powerful capabilities for searching and reasoning about structured records
9125 and relational data. Some even argue that the most important limitation of artificial intel-
9126 ligence is not inference or learning, but simply having too little knowledge (Lenat et al.,
9127 1990). Natural language processing provides an appealing solution: automatically con-
9128 struct a structured **knowledge base** by reading natural language text.

9129 For example, many Wikipedia pages have an “infobox” that provides structured in-
9130 formation about an entity or event. An example is shown in Figure 17.1a: each row rep-
9131 resents one or more properties of the entity IN THE AEROPLANE OVER THE SEA, a record
9132 album. The set of properties is determined by a predefined **schema**, which applies to all
9133 record albums in Wikipedia. As shown in Figure 17.1b, the values for many of these fields
9134 are indicated directly in the first few sentences of text on the same Wikipedia page.

9135 The task of automatically constructing (or “populating”) an infobox from text is an
9136 example of **information extraction**. Much of information extraction can be described in
9137 terms of **entities**, **relations**, and **events**.

- 9138 • **Entities** are uniquely specified objects in the world, such as people (JEFF MANGUM),
9139 places (ATHENS, GEORGIA), organizations (MERGE RECORDS), and times (FEBRUARY
9140 10, 1998). Chapter 8 described the task of **named entity recognition**, which labels
9141 tokens as parts of entity spans. Now we will see how to go further, **linking** each
9142 entity **mention** to an element in a **knowledge base**.
- 9143 • **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,...>
```

9144 The set of arguments for an event type is defined by a **schema**. Events often refer to
 9145 time-delimited occurrences: weddings, protests, purchases, terrorist attacks.

9146 Information extraction is similar to semantic role labeling (chapter 13): we may think
 9147 of predicates as corresponding to events, and the arguments as defining slots in the event
 9148 representation. However, the goals of information extraction are different. Rather than
 9149 accurately parsing every sentence, information extraction systems often focus on recog-
 9150 nizing a few key relation or event types, or on the task of identifying all properties of a
 9151 given entity. Information extraction is often evaluated by the correctness of the resulting
 9152 knowledge base, and not by how many sentences were accurately parsed. The goal is
 9153 sometimes described as **macro-reading**, as opposed to **micro-reading**, in which each sen-
 9154 tence must be analyzed correctly. Macro-reading systems are not penalized for ignoring
 9155 difficult sentences, as long as they can recover the same information from other, easier-
 9156 to-read sources. However, macro-reading systems must resolve apparent inconsistencies

9157 (was the album released on MERGE RECORDS or BLUE ROSE RECORDS?), requiring reasoning across the entire dataset.

9159 In addition to the basic tasks of recognizing entities, relations, and events, information
9160 extraction systems must handle negation, and must be able to distinguish statements of
9161 fact from hopes, fears, hunches, and hypotheticals. Finally, information extraction is often paired with the problem of **question answering**, which requires accurately parsing a
9163 query, and then selecting or generating a textual answer. Question answering systems can
9164 be built on knowledge bases that are extracted from large text corpora, or may attempt to
9165 identify answers directly from the source texts.

9166 17.1 Entities

9167 The starting point for information extraction is to identify mentions of entities in text.
9168 Consider the following example:

9169 (17.4) *The United States Army captured a hill overlooking Atlanta on May 14, 1864.*

9170 For this sentence, there are two goals:

- 9171 1. *Identify* the spans *United States Army*, *Atlanta*, and *May 14, 1864* as entity mentions.
9172 (The hill is not uniquely identified, so it is not a *named* entity.) We may also want to
9173 recognize the **named entity types**: organization, location, and date. This is **named**
9174 **entity recognition**, and is described in chapter 8.
- 9175 2. *Link* these spans to entities in a knowledge base: U.S. ARMY, ATLANTA, and 1864-
9176 MAY-14. This task is known as **entity linking**.

9177 The strings to be linked to entities are **mentions** — similar to the use of this term in
9178 coreference resolution. In some formulations of the entity linking task, only named entities
9179 are candidates for linking. This is sometimes called **named entity linking** (Ling et al.,
9180 2015). In other formulations, such as **Wikification** (Milne and Witten, 2008), any string
9181 can be a mention. The set of target entities often corresponds to Wikipedia pages, and
9182 Wikipedia is the basis for more comprehensive knowledge bases such as YAGO (Suchanek
9183 et al., 2007), DBPedia (Auer et al., 2007), and Freebase (Bollacker et al., 2008). Entity link-
9184 ing may also be performed in more “closed” settings, where a much smaller list of targets
9185 is provided in advance. The system must also determine if a mention does not refer to
9186 any entity in the knowledge base, sometimes called a **NIL entity** (McNamee and Dang,
9187 2009).

9188 Returning to (17.4), the three entity mentions may seem unambiguous. But the Wikipedia
9189 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.

9190 different towns and cities, five United States Navy vessels, a magazine, a television show,
 9191 a band, and a singer — each prominent enough to have its own Wikipedia page. We now
 9192 consider how to choose among these dozens of possibilities. In this chapter we will focus
 9193 on supervised approaches. Unsupervised entity linking is closely related to the problem
 9194 of **cross-document coreference resolution**, where the task is to identify pairs of mentions
 9195 that corefer, across document boundaries (Bagga and Baldwin, 1998b; Singh et al., 2011).

9196 **17.1.1 Entity linking by learning to rank**

9197 Entity linking is often formulated as a **ranking** problem,

$$\hat{y} = \underset{y \in \mathcal{Y}(x)}{\operatorname{argmax}} \Psi(y, x, c), \quad [17.1]$$

9198 where y is a target entity, x is a description of the mention, $\mathcal{Y}(x)$ is a set of candidate
 9199 entities, and c is a description of the context — such as the other text in the document,
 9200 or its metadata. The function Ψ is a scoring function, which could be a linear model,
 9201 $\Psi(y, x, c) = \theta \cdot f(y, x, c)$, or a more complex function such as a neural network. In either
 9202 case, the scoring function can be learned by minimizing a margin-based **ranking loss**,

$$\ell(\hat{y}, y^{(i)}, x^{(i)}, c^{(i)}) = \left(\Psi(\hat{y}, x^{(i)}, c^{(i)}) - \Psi(y^{(i)}, x^{(i)}, c^{(i)}) + 1 \right)_+, \quad [17.2]$$

9203 where $y^{(i)}$ is the ground truth and $\hat{y} \neq y^{(i)}$ is the predicted target for mention $x^{(i)}$ in
 9204 context $c^{(i)}$ (Joachims, 2002; Dredze et al., 2010).

9205 **Candidate identification** For computational tractability, it is helpful to restrict the set of
 9206 candidates, $\mathcal{Y}(x)$. One approach is to use a **name dictionary**, which maps from strings
 9207 to the entities that they might mention. This mapping is many-to-many: a string such as
 9208 *Atlanta* can refer to multiple entities, and conversely, an entity such as ATLANTA can be
 9209 referenced by multiple strings. A name dictionary can be extracted from Wikipedia, with
 9210 links between each Wikipedia entity page and the anchor text of all hyperlinks that point
 9211 to the page (Bunescu and Pasca, 2006; Ratinov et al., 2011). To improve recall, the name
 9212 dictionary can be augmented by partial and approximate matching (Dredze et al., 2010),
 9213 but as the set of candidates grows, the risk of false positives increases. For example, the
 9214 string *Atlanta* is a partial match to *the Atlanta Fed* (a name for the FEDERAL RESERVE BANK
 9215 OF ATLANTA), and a noisy match (edit distance of one) from *Atalanta* (a heroine in Greek
 9216 mythology and an Italian soccer team).

9217 **Features** Feature-based approaches to entity ranking rely on three main types of local
 9218 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).

- 9249 on the text of Wikipedia, with hyperlinks substituted for the anchor text.³
- 9250 • The embedding of the mention x can be computed by averaging the embeddings
 9251 of the words in the mention (Yang et al., 2016), or by the compositional techniques
 9252 described in § 14.8.
- 9253 • The embedding of the context c can also be computed from the embeddings of the
 9254 words in the context. A **denoising autoencoder** learns a function from raw text to
 9255 dense K -dimensional vector encodings by minimizing a reconstruction loss (Vin-
 9256 cent et al., 2010),

$$\min_{\theta_g, \theta_h} \sum_{i=1}^N \|\mathbf{x}^{(i)} - g(h(\tilde{\mathbf{x}}^{(i)}; \theta_h); \theta_g)\|^2, \quad [17.4]$$

9253 where $\tilde{\mathbf{x}}^{(i)}$ is a noisy version of the bag-of-words counts $\mathbf{x}^{(i)}$, which is produced by
 9254 randomly setting some counts to zero; $h : \mathbb{R}^V \rightarrow \mathbb{R}^K$ is an encoder with parameters
 9255 θ_h ; and $g : \mathbb{R}^K \rightarrow \mathbb{R}^V$, with parameters θ_g . The encoder and decoder functions
 9256 are typically implemented as feedforward neural networks. To apply this model to
 9257 entity linking, each entity and context are initially represented by the encoding of
 9258 their bag-of-words vectors, $h(e)$ and $g(c)$, and these encodings are then fine-tuned
 9259 from labeled data (He et al., 2013). The context vector c can also be obtained by
 9260 convolution on the embeddings of words in the document (Sun et al., 2015), or by
 9261 examining metadata such as the author’s social network (Yang et al., 2016).

9262 The remaining parameters $\Theta^{(y,x)}$ and $\Theta^{(y,c)}$ can be trained by backpropagation from the
 9263 margin loss in Equation 17.2.

9264 17.1.2 Collective entity linking

9265 Entity linking can be more accurate when it is performed jointly across a document. To
 9266 see why, consider the following lists:

- 9267 (17.5) California, Oregon, Washington
 9268 (17.6) Baltimore, Washington, Philadelphia
 9269 (17.7) Washington, Adams, Jefferson

9270 In each case, the term *Washington* refers to a different entity, and this reference is strongly
 9271 suggested by the other entries on the list. In the last list, all three names are highly am-
 9272 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/>.

9273 preference for coherence motivates **collectively** linking these references to the first three
 9274 U.S. presidents.

9275 A general approach to collective entity linking is to introduce a compatibility score
 9276 $\psi_c(\mathbf{y})$. Collective entity linking is then performed by optimizing the global objective,

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathbb{Y}(\mathbf{x})}{\operatorname{argmax}} \Psi_c(\mathbf{y}) + \sum_{i=1}^N \Psi_\ell(y^{(i)}, \mathbf{x}^{(i)}, \mathbf{c}^{(i)}), \quad [17.5]$$

9277 where $\mathbb{Y}(\mathbf{x})$ is the set of all possible collective entity assignments for the mentions in \mathbf{x} ,
 9278 and ψ_ℓ is the local scoring function for each entity i . The compatibility function is typically
 9279 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
 9280 can be computed in a number of different ways:

- 9281 • Wikipedia defines high-level categories for entities (e.g., *living people*, *Presidents of*
 9282 *the United States*, *States of the United States*), and Ψ_c can reward entity pairs for the
 9283 number of categories that they have in common (Cucerzan, 2007).
- 9284 • Compatibility can be measured by the number of incoming hyperlinks shared by
 9285 the Wikipedia pages for the two entities (Milne and Witten, 2008).
- 9286 • In a neural architecture, the compatibility of two entities can be set equal to the inner
 9287 product of their embeddings, $\Psi_c(y^{(i)}, y^{(j)}) = \mathbf{v}_{y^{(i)}} \cdot \mathbf{v}_{y^{(j)}}$.
- 9288 • A non-pairwise compatibility score can be defined using a type of latent variable
 9289 model known as a **probabilistic topic model** (Blei et al., 2003; Blei, 2012). In this
 9290 framework, each latent topic is a probability distribution over entities, and each
 9291 document has a probability distribution over topics. Each entity helps to determine
 9292 the document's distribution over topics, and in turn these topics help to resolve am-
 9293 biguous entity mentions (Newman et al., 2006). Inference can be performed using
 9294 the sampling techniques described in chapter 5.

9295 Unfortunately, collective entity linking is **NP-hard** even for pairwise compatibility func-
 9296 tions, so exact optimization is almost certainly intractable. Various approximate inference
 9297 techniques have been proposed, including **integer linear programming** (Cheng and Roth,
 9298 2013), **Gibbs sampling** (Han and Sun, 2012), and graph-based algorithms (Hoffart et al.,
 9299 2011; Han et al., 2011).

9300 17.1.3 *Pairwise ranking loss functions

9301 The loss function defined in Equation 17.2 considers only the highest-scoring prediction
 9302 \hat{y} , but in fact, the true entity $y^{(i)}$ should outscore *all* other entities. A loss function based on
 9303 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

```

9304 not just the top-scoring prediction. Usunier et al. (2009) define a general ranking error
 9305 function,

$$L_{\text{rank}}(k) = \sum_{j=1}^k \alpha_j, \quad \text{with } \alpha_1 \geq \alpha_2 \geq \dots \geq 0, \quad [17.6]$$

9306 where k is equal to the number of labels ranked higher than the correct label $y^{(i)}$. This
 9307 function defines a class of ranking errors: if $\alpha_j = 1$ for all j , then the ranking error is
 9308 equal to the rank of the correct entity; if $\alpha_1 = 1$ and $\alpha_{j>1} = 0$, then the ranking error is
 9309 one whenever the correct entity is not ranked first; if α_j decreases smoothly with j , as in
 9310 $\alpha_j = \frac{1}{j}$, 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,

$$r(y^{(i)}, \mathbf{x}^{(i)}) \triangleq \sum_{y \in \mathcal{Y}(\mathbf{x}^{(i)}) \setminus y^{(i)}} \delta \left(1 + \psi(y, \mathbf{x}^{(i)}) \geq \psi(y^{(i)}, \mathbf{x}^{(i)}) \right), \quad [17.7]$$

9311 where $\delta(\cdot)$ is a delta function, and $\mathcal{Y}(\mathbf{x}^{(i)}) \setminus y^{(i)}$ is the set of all entity candidates minus
 9312 the true entity $y^{(i)}$. The margin-augmented rank is the rank of the true entity, after aug-
 9313 menting every other candidate with a margin of one, under the current scoring function
 9314 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,

$$\ell(y^{(i)}, \mathbf{x}^{(i)}) = \frac{L_{\text{rank}}(r(y^{(i)}, \mathbf{x}^{(i)}))}{r(y^{(i)}, \mathbf{x}^{(i)})} \sum_{y \in \mathcal{Y}(\mathbf{x}) \setminus y^{(i)}} \left(\psi(y, \mathbf{x}^{(i)}) - \psi(y^{(i)}, \mathbf{x}^{(i)}) + 1 \right)_+, \quad [17.8]$$

The sum in Equation 17.8 includes non-zero values for every label that is ranked at least as high as the true entity, after applying the margin augmentation. Dividing by the margin-augmented rank of the true entity thus gives the average violation.

The objective in Equation 17.8 is expensive to optimize when the label space is large, as is usually the case for entity linking against large knowledge bases. This motivates a randomized approximation called **WARP** (Weston et al., 2011), shown in Algorithm 19. In this procedure, we sample random entities until one violates the pairwise margin constraint, $\psi(y, \mathbf{x}^{(i)}) + 1 \geq \psi(y^{(i)}, \mathbf{x}^{(i)})$. The number of samples N required to find such a violation yields an approximation of the margin-augmented rank of the true entity, $r(y^{(i)}, \mathbf{x}^{(i)}) \approx \left\lceil \frac{|\mathcal{Y}(\mathbf{x})|}{N} \right\rceil$. If a violation is found immediately, $N = 1$, the correct entity probably ranks below many others, $r \approx |\mathcal{Y}(\mathbf{x})|$. If many samples are required before a violation is found, $N \rightarrow |\mathcal{Y}(\mathbf{x})|$, then the correct entity is probably highly ranked, $r \rightarrow 1$. A computational advantage of WARP is that it is not necessary to find the highest-scoring label, which can impose a non-trivial computational cost when $\mathcal{Y}(\mathbf{x}^{(i)})$ is large. The objective is conceptually similar to the **negative sampling** objective in WORD2VEC (chapter 14), which compares the observed word against randomly sampled alternatives.

17.2 Relations

After identifying the entities that are mentioned in a text, the next step is to determine how they are related. Consider the following example:

(17.8) George Bush traveled to France on Thursday for a summit.

This sentence introduces a relation between the entities referenced by *George Bush* and *France*. In the Automatic Content Extraction (ACE) ontology (Linguistic Data Consortium, 2005), the type of this relation is PHYSICAL, and the subtype is LOCATED. This relation would be written,

$$\text{PHYSICAL.LOCATED(GEORGE BUSH, FRANCE).} \quad [17.9]$$

Relations take exactly two arguments, and the order of the arguments matters.

In the ACE datasets, relations are annotated between entity mentions, as in the example above. Relations can also hold between nominals, as in the following example from 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)

9343 (17.9) The cup contained tea from dried ginseng.

9344 This sentence describes a relation of type ENTITY-ORIGIN between *tea* and *ginseng*. Nominal
 9345 relation extraction is closely related to **semantic role labeling** (chapter 13). The main
 9346 difference is that relation extraction is restricted to a relatively small number of relation
 9347 types; for example, Table 17.1 shows the ten relation types from SemEval-2010.

9348 17.2.1 Pattern-based relation extraction

9349 Early work on relation extraction focused on hand-crafted patterns (Hearst, 1992). For
 9350 example, the appositive *Starbuck, a native of Nantucket* signals the relation ENTITY-ORIGIN
 9351 between *Starbuck* and *Nantucket*. This pattern can be written as,

$$\text{PERSON , } a \text{ native of LOCATION} \Rightarrow \text{ENTITY-ORIGIN(PERSON, LOCATION)}. \quad [17.10]$$

9352 This pattern will be “triggered” whenever the literal string *, a native of* occurs between an
 9353 entity of type PERSON and an entity of type LOCATION. Such patterns can be generalized
 9354 beyond literal matches using techniques such as lemmatization, which would enable the
 9355 words (*buy, buys, buying*) to trigger the same patterns (see § 4.3.1.2). A more aggressive
 9356 strategy would be to group all words in a WordNet synset (§ 4.2), so that, e.g., *buy* and
 9357 *purchase* trigger the same patterns.

9358 Relation extraction patterns can be implemented in finite-state automata (§ 9.1). If the
 9359 named entity recognizer is also a finite-state machine, then the systems can be combined
 9360 by finite-state transduction (Hobbs et al., 1997). This makes it possible to propagate uncer-
 9361 tainty through the finite-state cascade, and disambiguate from higher-level context. For
 9362 example, suppose the entity recognizer cannot decide whether *Starbuck* refers to either a
 9363 PERSON or a LOCATION; in the composed transducer, the relation extractor would be free
 9364 to select the PERSON annotation when it appears in the context of an appropriate pattern.

9365 **17.2.2 Relation extraction as a classification task**

9366 Relation extraction can be formulated as a classification problem,

$$\hat{r}_{(i,j),(m,n)} = \operatorname{argmax}_{r \in \mathcal{R}} \Psi(r, (i, j), (m, n), \mathbf{w}), \quad [17.11]$$

9367 where $r \in \mathcal{R}$ is a relation type (possibly NIL), $\mathbf{w}_{i+1:j}$ is the span of the first argument, and
 9368 $\mathbf{w}_{m+1:n}$ is the span of the second argument. The argument $\mathbf{w}_{m+1:n}$ may appear before
 9369 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
 9370 argument of the relation. We now consider three alternatives for computing the scoring
 9371 function.

9372 **17.2.2.1 Feature-based classification**

9373 In a feature-based classifier, the scoring function is defined as,

$$\Psi(r, (i, j), (m, n), \mathbf{w}) = \boldsymbol{\theta} \cdot \mathbf{f}(r, (i, j), (m, n), \mathbf{w}), \quad [17.12]$$

9374 with $\boldsymbol{\theta}$ representing a vector of weights, and $\mathbf{f}(\cdot)$ a vector of features. The pattern-based
 9375 methods described in § 17.2.1 suggest several features:

- 9376 • Local features of $\mathbf{w}_{i+1:j}$ and $\mathbf{w}_{m+1:n}$, including: the strings themselves; whether they
 9377 are recognized as entities, and if so, which type; whether the strings are present in a
 9378 **gazetteer** of entity names; each string's syntactic **head** (§ 9.2.2).
- 9379 • Features of the span between the two arguments, $\mathbf{w}_{j+1:m}$ or $\mathbf{w}_{n+1:i}$ (depending on
 9380 which argument appears first): the length of the span; the specific words that appear
 9381 in the span, either as a literal sequence or a bag-of-words; the wordnet synsets (§ 4.2)
 9382 that appear in the span between the arguments.
- 9383 • Features of the syntactic relationship between the two arguments, typically the **de-**
 9384 **pendency path** between the arguments (§ 13.2.1). Example dependency paths are
 9385 shown in Table 17.2.

9386 **17.2.2.2 Kernels**

9387 Suppose that the first line of Table 17.2 is a labeled example, and the remaining lines are
 9388 instances to be classified. A feature-based approach would have to decompose the depen-
 9389 dency paths into features that capture individual edges, with or without their labels, and
 9390 then learn weights for each of these features: for example, the second line contains identi-
 9391 cal dependencies, but different arguments; the third line contains a different inflection of
 9392 the word *travel*; the fourth and fifth lines each contain an additional edge on the depen-
 9393 dency path; and the sixth example uses an entirely different path. Rather than attempting
 9394 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 traveled \rightarrow France NSUBJ OBL
2. <i>Ahab traveled to Nantucket</i>	<i>Ahab</i> \leftarrow traveled \rightarrow Nantucket NSUBJ OBL
3. <i>George Bush will travel to France</i>	<i>George Bush</i> \leftarrow travel \rightarrow France NSUBJ OBL
4. <i>George Bush wants to travel to France</i>	<i>George Bush</i> \leftarrow wants \rightarrow travel \rightarrow France NSUBJ XCOMP OBL
5. <i>Ahab traveled to a city in France</i>	<i>Ahab</i> \leftarrow traveled \rightarrow city \rightarrow France NSUBJ OBL NMOD
6. <i>We await Ahab's visit to France</i>	<i>Ahab</i> \leftarrow visit \rightarrow France NMOD:POSS NMOD

Table 17.2: Candidates instances for the PHYSICAL.LOCATED relation, and their dependency paths

9395 are similar and different, we can instead define a similarity function κ , which computes a
 9396 score for any pair of instances, $\kappa : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_+$. The score for any pair of instances (i, j)
 9397 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
 9398 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,

$$\hat{y} = \text{Sign}(b + \sum_{i=1}^N y^{(i)} \alpha^{(i)} \kappa(\mathbf{x}^{(i)}, \mathbf{x})) \quad [17.13]$$

9399 where b and $\{\alpha^{(i)}\}_{i=1}^N$ are parameters that must be learned from the training set, under
 9400 the constraint $\forall_i, \alpha^{(i)} \geq 0$. Intuitively, each α_i specifies the importance of the instance $\mathbf{x}^{(i)}$
 9401 towards the classification rule. Kernel-based classification can be viewed as a weighted
 9402 form of the **nearest-neighbor** classifier (Hastie et al., 2009), in which test instances are
 9403 assigned the most common label among their near neighbors in the training set. This
 9404 results in a non-linear classification boundary. The parameters are typically learned from
 9405 a margin-based objective (see § 2.3), leading to the **kernel support vector machine**. To
 9406 generalize to multi-class classification, we can train separate binary classifiers for each
 9407 label (sometimes called **one-versus-all**), or train binary classifiers for each pair of possible
 9408 labels (**one-versus-one**).

9409 Dependency kernels are particularly effective for relation extraction, due to their ability
 9410 to capture syntactic properties of the path between the two candidate arguments. One
 9411 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,

9438 (17.10) *George Bush* $\xleftarrow[\text{NSUBJ}]{} \text{wants}$

9439 (17.11) $\text{wants} \xrightarrow[\text{XCOMP}]{} \text{travel} \xrightarrow[\text{OBL}]{} \text{France}.$

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 $\mathbf{x}_m^{(c)}$ indicate the embedding of word (or relation) m in channel c , and let $\overleftarrow{\mathbf{h}}_m^{(c)}$ and $\overrightarrow{\mathbf{h}}_m^{(c)}$ indicate the associated recurrent states in the left and right subtrees respectively. Then the complete model is specified as follows,

$$\mathbf{h}_{s(m)}^{(c)} = \text{RNN}(\mathbf{x}_{s(m)}^{(c)}, \mathbf{h}_m^{(c)}) \quad [17.18]$$

$$\mathbf{z}^{(c)} = \text{MaxPool}(\overleftarrow{\mathbf{h}}_i^{(c)}, \overleftarrow{\mathbf{h}}_{s(i)}^{(c)}, \dots, \overleftarrow{\mathbf{h}}_{\text{root}}^{(c)}, \overrightarrow{\mathbf{h}}_j^{(c)}, \overrightarrow{\mathbf{h}}_{s(j)}^{(c)}, \dots, \overrightarrow{\mathbf{h}}_{\text{root}}^{(c)}) \quad [17.19]$$

$$\Psi(r, i, j) = \theta \cdot [\mathbf{z}^{(\text{word})}; \mathbf{z}^{(\text{POS})}; \mathbf{z}^{(\text{dependency})}; \mathbf{z}^{(\text{hypernym})}]. \quad [17.20]$$

9440 Note that \mathbf{z} is computed by applying max-pooling to the *matrix* of horizontally concatenated vectors \mathbf{h} , while Ψ is computed from the *vector* of vertically concatenated vectors
 9441 \mathbf{z} . Xu et al. (2015) pass the score Ψ through a **softmax** layer to obtain a probability
 9442 $p(r | i, j, w)$, and train the model by regularized **cross-entropy**. Miwa and Bansal (2016)
 9443 show that a related model can solve the more challenging “end-to-end” relation extraction
 9444 task, in which the model must simultaneously detect entities and then extract their
 9445 relations.
 9446

9447 17.2.3 Knowledge base population

9448 In many applications, what matters is not what fraction of sentences are analyzed cor-
 9449 rectly, but how much accurate knowledge can be extracted. **Knowledge base population**
 9450 (**KBP**) refers to the task of filling in Wikipedia-style infoboxes, as shown in Figure 17.1a.
 9451 Knowledge base population can be decomposed into two subtasks: **entity linking** (de-
 9452 scribed in § 17.1), and **slot filling** (Ji and Grishman, 2011). Slot filling has two key dif-
 9453 ferences from the formulation of relation extraction presented above: the relations hold
 9454 between entities rather than spans of text, and the performance is evaluated at the *type*
 9455 *level* (on entity pairs), rather than on the *token level* (on individual sentences).

9456 From a practical standpoint, there are three other important differences between slot
 9457 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

9489 One approach is to run a single-document relation extraction system (using the tech-
 9490 niques described in § 17.2.2), and then aggregate the results (Li et al., 2011). Relations
 9491 that are detected with high confidence in multiple documents are more likely to be valid,
 9492 motivating the heuristic,

$$\psi(r, e_1, e_2) = \sum_{i=1}^N (\text{p}(r(e_1, e_2) | \mathbf{w}^{(i)}))^\alpha, \quad [17.21]$$

9493 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-
 9494 tioned on the text $\mathbf{w}^{(i)}$, and $\alpha \gg 1$ is a tunable hyperparameter. Using this heuristic, it is
 9495 possible to rank all candidate relations, and trace out a **precision-recall curve** as more rel-
 9496ations are extracted.⁶ Alternatively, features can be aggregated across multiple passages
 9497 of text, feeding a single type-level relation extraction system (Wolfe et al., 2017).

9498 Precision can be improved by introducing constraints across multiple relations. For
 9499 example, if we are certain of the relation $\text{PARENT}(e_1, e_2)$, then it cannot also be the case
 9500 that $\text{PARENT}(e_2, e_1)$. Integer linear programming makes it possible to incorporate such
 9501 constraints into a global optimization (Li et al., 2011). Other pairs of relations have pos-
 9502 tive correlations, such $\text{MAYOR}(e_1, e_2)$ and $\text{LIVED-IN}(e_1, e_2)$. Compatibility across relation
 9503 types can be incorporated into probabilistic graphical models (e.g., Riedel et al., 2010).

9504 17.2.3.2 Distant supervision

9505 Relation extraction is “annotation hungry,” because each relation requires its own la-
 9506 beled data. Rather than relying on annotations of individual documents, it would be
 9507 preferable to use existing knowledge resources — such as the many facts that are al-
 9508 ready captured in knowledge bases like DBpedia. However such annotations raise the
 9509 inverse of the information fusion problem considered above: the existence of the relation
 9510 $\text{MAYOR}(\text{MAYNARD JACKSON JR., ATLANTA})$ provides only **distant supervision** for the
 9511 example texts in which this entity pair is mentioned.

9512 One approach is to treat the entity pair as the instance, rather than the text itself (Mintz
 9513 et al., 2009). Features are then aggregated across all sentences in which both entities are
 9514 mentioned, and labels correspond to the relation (if any) between the entities in a knowl-
 9515 edge base, such as FreeBase. Negative instances are constructed from entity pairs that are
 9516 not related in the knowledge base. In some cases, two entities are related, but the knowl-
 9517 edge base is missing the relation; however, because the number of possible entity pairs is
 9518 huge, these missing relations are presumed to be relatively rare. This approach is shown
 9519 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.

9520 In **multiple instance learning**, labels are assigned to *sets* of instances, of which only
 9521 an unknown subset are actually relevant (Dietterich et al., 1997; Maron and Lozano-Pérez,
 9522 1998). This formalizes the framework of distant supervision: the relation $\text{REL}(A, B)$ acts
 9523 as a label for the entire set of sentences mentioning entities A and B, even when only a
 9524 subset of these sentences actually describes the relation. One approach to multi-instance
 9525 learning is to introduce a binary **latent variable** for each sentence, indicating whether the
 9526 sentence expresses the labeled relation (Riedel et al., 2010). A variety of inference tech-
 9527 niques have been employed for this probabilistic model of relation extraction: Surdeanu
 9528 et al. (2012) use expectation maximization, Riedel et al. (2010) use sampling, and Hoff-
 9529 mann et al. (2011) use a custom graph-based algorithm. Expectation maximization and
 9530 sampling are surveyed in chapter 5, and are covered in more detail by Murphy (2012);
 9531 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.

9532 17.2.4 Open information extraction

9533 In classical relation extraction, the set of relations is defined in advance, using a **schema**.
 9534 The relation for any pair of entities can then be predicted using multi-class classification.
 9535 In **open information extraction** (OpenIE), a relation can be any triple of text. The example
 9536 sentence (17.12) instantiates several “relations” of this sort, e.g.,

- 9537 • (*mayor of, Maynard Jackson, Atlanta*),
- 9538 • (*elected, Maynard Jackson, mayor of Atlanta*),
- 9539 • (*elected in, Maynard Jackson, 1973*).

9540 Extracting such tuples can be viewed as a lightweight version of **semantic role labeling**
 9541 (chapter 13), with only two argument types: first slot and second slot. The task is gen-
 9542 erally evaluated on the relation level, rather than on the level of sentences: precision is
 9543 measured by the number of extracted relations that are accurate, and recall is measured
 9544 by the number of true relations that were successfully extracted. OpenIE systems are
 9545 trained from distant supervision or bootstrapping, rather than from labeled sentences.

9546 An early example is the `TextRunner` system (Banko et al., 2007), which identifies re-
 9547 lations with a set of handcrafted syntactic rules. The examples that are acquired from the
 9548 handcrafted rules are then used to train a classification model that uses part-of-speech pat-
 9549 terns as features. Finally, the relations that are extracted by the classifier are aggregated,
 9550 removing redundant relations and computing the number of times that each relation is
 9551 mentioned in the corpus. `TextRunner` was the first in a series of systems that performed
 9552 increasingly accurate open relation extraction by incorporating more precise linguistic fea-
 9553 tures (Etzioni et al., 2011), distant supervision from Wikipedia infoboxes (Wu and Weld,
 9554 2010), and better learning algorithms (Zhu et al., 2009).

9555 17.3 Events

9556 Relations link pairs of entities, but many real-world situations involve more than two en-
9557 tities. Consider again the example sentence (17.12), which describes the **event** of an elec-
9558 tion, with four properties: the office (MAYOR), the district (ATLANTA), the date (1973), and
9559 the person elected (MAYNARD JACKSON, JR.). In **event detection**, a schema is provided
9560 for each event type (e.g., an election, a terrorist attack, or a chemical reaction), indicating
9561 all the possible properties of the event. The system is then required to fill in as many of
9562 these properties as possible (Doddington et al., 2004).

9563 Event detection systems generally involve a retrieval component (finding relevant
9564 documents and passages of text) and an extraction component (determining the proper-
9565 ties of the event based on the retrieved texts). Early approaches focused on finite-state pat-
9566 terns for identify event properties (Hobbs et al., 1997); such patterns can be automatically
9567 induced by searching for patterns that are especially likely to appear in documents that
9568 match the event query (Riloff, 1996). Contemporary approaches employ techniques that
9569 are similar to FrameNet semantic role labeling (§ 13.2), such as structured prediction over
9570 local and global features (Li et al., 2013) and bidirectional recurrent neural networks (Feng
9571 et al., 2016). These methods detect whether an event is described in a sentence, and if so,
9572 what are its properties.

9573 **Event coreference** Because multiple sentences may describe unique properties of a sin-
9574 gle event, **event coreference** is required to link event mentions across a single passage
9575 of text, or between passages (Humphreys et al., 1997). Bejan and Harabagiu (2014) de-
9576 fine event coreference as the task of identifying event mentions that share the same event
9577 participants (i.e., the slot-filling entities) and the same event properties (e.g., the time and
9578 location), within or across documents. Event coreference resolution can be performed us-
9579 ing supervised learning techniques in a similar way to entity coreference, as described
9580 in chapter 15: move left-to-right through the document, and use a classifier to decide
9581 whether to link each event reference to an existing cluster of coreferent events, or to cre-
9582 ate a new cluster (Ahn, 2006). Each clustering decision is based on the compatibility of
9583 features describing the participants and properties of the event. Due to the difficulty of
9584 annotating large amounts of data for entity coreference, unsupervised approaches are es-
9585 pecially desirable (Chen and Ji, 2009; Bejan and Harabagiu, 2014).

9586 **Relations between events** Just as entities are related to other entities, events may be
9587 related to other events: for example, the event of winning an election both *precedes* and
9588 *causes* the event of serving as mayor; moving to Atlanta *precedes* and *enables* the event of
9589 becoming mayor of Atlanta; moving from Dallas to Atlanta *prevents* the event of later be-
9590 coming mayor of Dallas. As these examples show, events may be related both temporally
9591 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.

9592 between events (Pustejovsky et al., 2005), derived in part from **interval algebra** (Allen,
9593 1984). The TimeBank corpus provides TimeML annotations for 186 documents (Pustejovsky
9594 et al., 2003). Methods for detecting these temporal relations combine supervised
9595 machine learning with temporal constraints, such as transitivity (e.g. Mani et al., 2006;
9596 Chambers and Jurafsky, 2008).

9597 More recent annotation schemes and datasets combine temporal and causal relations (Mirza
9598 et al., 2014; Dunietz et al., 2017): for example, the CaTeRS dataset includes annotations of
9599 320 five-sentence short stories (Mostafazadeh et al., 2016). Abstracting still further, **processes**
9600 are networks of causal relations between multiple events. A small dataset of biological
9601 processes is annotated in the ProcessBank dataset (Berant et al., 2014), with the
9602 goal of supporting automatic question answering on scientific textbooks.

9603 17.4 Hedges, denials, and hypotheticals

9604 The methods described thus far apply to **propositions** about the way things are in the
9605 real world. But natural language can also describe events and relations that are likely or
9606 unlikely, possible or impossible, desired or feared. The following examples hint at the
9607 scope of the problem (Prabhakaran et al., 2010):

- 9608 (17.16) GM will lay off workers.
- 9609 (17.17) A spokesman for GM said GM will lay off workers.
- 9610 (17.18) GM may lay off workers.
- 9611 (17.19) The politician claimed that GM will lay off workers.
- 9612 (17.20) Some wish GM would lay off workers.
- 9613 (17.21) Will GM lay off workers?
- 9614 (17.22) Many wonder whether GM will lay off workers.

9615 Accurate information extraction requires handling these **extra-propositional** aspects
9616 of meaning, which are sometimes summarized under the terms **modality** and **negation**.⁷
9617 Modality refers to expressions of the speaker's attitude towards her own statements, in-
9618 cluding "degree of certainty, reliability, subjectivity, sources of information, and perspec-
9619 tive" (Morante and Sporleder, 2012). Various systematizations of modality have been
9620 proposed (e.g., Palmer, 2001), including categories such as future, interrogative, imper-
9621 ative, conditional, and subjective. Information extraction is particularly concerned with
9622 negation and certainty. For example, Saurí and Pustejovsky (2009) link negation with
9623 a modal calculus of certainty, likelihood, and possibility, creating the two-dimensional
9624 schema shown in Table 17.4. This is the basis for the FactBank corpus, with annotations
9625 of the **factuality** of all sentences in 208 documents of news text.

9626 A related concept is **hedging**, in which speakers limit their commitment to a proposi-
9627 tion (Lakoff, 1973):

- 9628 (17.23) These results **suggest** that expression of c-jun, jun B and jun D genes **might** be in-
9629 volved in terminal granulocyte differentiation... (Morante and Daelemans, 2009)
9630 (17.24) A whale is **technically** a mammal (Lakoff, 1973)

9631 In the first example, the hedges *suggest* and *might* communicate uncertainty; in the second
9632 example, there is no uncertainty, but the hedge *technically* indicates that the evidence for
9633 the proposition will not fully meet the reader's expectations. Hedging has been studied
9634 extensively in scientific texts (Medlock and Briscoe, 2007; Morante and Daelemans, 2009),
9635 where the goal of large-scale extraction of scientific facts is obstructed by hedges and spec-
9636 ulation. Still another related aspect of modality is **evidentiality**, in which speakers mark
9637 the source of their information. In many languages, it is obligatory to mark evidentiality
9638 through affixes or particles (Aikhenvald, 2004); while evidentiality is not grammaticalized
9639 in English, authors are expected to express this information in contexts such as journal-
9640 ism (Kovach and Rosenstiel, 2014) and Wikipedia.⁸

9641 Methods for handling negation and modality generally include two phases:

- 9642 1. detecting negated or uncertain events;
9643 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>

9644 A considerable body of work on negation has employed rule-based techniques such as
 9645 regular expressions (Chapman et al., 2001) to detect negated events. Such techniques
 9646 match lexical cues (e.g., *Norwood was not elected Mayor*), while avoiding “double nega-
 9647 tives” (e.g., *surely all this is not without meaning*). More recent approaches employ classi-
 9648 fiers over lexical and syntactic features (Uzuner et al., 2009) and sequence labeling (Prab-
 9649 hakaran et al., 2010).

9650 The tasks of scope and focus resolution are more fine grained, as shown in the example
 9651 from Morante and Sporleder (2012):

- 9652 (17.25) [After his habit he said] **nothing**, and after mine I asked no questions.
 9653 After his habit he said nothing, and [after mine I asked] **no** [questions].

9654 In this sentence, there are two negation cues (*nothing* and *no*). Each negates an event,
 9655 indicated by the underlined verbs *said* and *asked* (this is the focus of negation), and each
 9656 occurs within a scope: *after his habit he said* and *after mine I asked* ____ *questions*. These tasks
 9657 are typically formalized as sequence labeling problems, with each word token labeled
 9658 as beginning, inside, or outside of a cue, focus, or scope span (see § 8.3). Conventional
 9659 sequence labeling approaches can then be applied, using surface features as well as syn-
 9660 tax (Velldal et al., 2012) and semantic analysis (Packard et al., 2014). Labeled datasets
 9661 include the BioScope corpus of biomedical texts (Vincze et al., 2008) and a shared task
 9662 dataset of detective stories by Arthur Conan Doyle (Morante and Blanco, 2012).

9663 17.5 Question answering and machine reading

9664 The victory of the Watson question-answering system against three top human players on
 9665 the game show *Jeopardy!* was a landmark moment for natural language processing (Fer-
 9666 rucci et al., 2010). Game show questions are usually answered by **factoids**: entity names
 9667 and short phrases.⁹ The task of factoid question answering is therefore closely related to
 9668 information extraction, with the additional problem of accurately parsing the question.

9669 17.5.1 Formal semantics

9670 Semantic parsing is an effective method for question-answering in restricted domains
 9671 such as questions about geography and airline reservations (Zettlemoyer and Collins,
 9672 2005), and has also been applied in “open-domain” settings such as question answering
 9673 on Freebase (Berant et al., 2013) and biomedical research abstracts (Poon and Domingos,
 9674 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).

9675 returns a boolean value: for example, the question *who is the mayor of the capital of Georgia?*
 9676 would be converted to,

$$\lambda x. \exists y \text{ CAPITAL(GEORGIA, } y) \wedge \text{MAYOR}(y, x). \quad [17.22]$$

9677 This lambda expression can then be used to query an existing knowledge base, returning
 9678 “true” for all entities that satisfy it.

9679 17.5.2 Machine reading

9680 Recent work has focused on answering questions about specific textual passages, similar
 9681 to the reading comprehension examinations for young students (Hirschman et al., 1999).
 9682 This task has come to be known as **machine reading**.

9683 17.5.2.1 Datasets

9684 The machine reading problem can be formulated in a number of different ways. The most
 9685 important distinction is what form the answer should take.

- 9686 • **Multiple-choice question answering**, as in the MCTest dataset of stories (Richardson et al., 2013) and the New York Regents Science Exams (Clark, 2015). In MCTest,
 9687 the answer is deducible from the text alone, while in the science exams, the system
 9688 must make inferences using an existing model of the underlying scientific phenomena.
 9689 Here is an example from MCTest:

9691 (17.26) James the turtle was always getting into trouble. Sometimes he'd reach into
 9692 the freezer and empty out all the food ...

9693 Q: What is the name of the trouble making turtle?
 9694 (a) Fries
 9695 (b) Pudding
 9696 (c) James
 9697 (d) Jane

- 9698 • **Cloze-style “fill in the blank”** questions, as in the CNN/Daily Mail comprehension
 9699 task (Hermann et al., 2015), the Children’s Book Test (Hill et al., 2016), and the Who-
 9700 did-What dataset (Onishi et al., 2016). In these tasks, the system must guess which
 9701 word or entity completes a sentence, based on reading a passage of text. Here is an
 9702 example from Who-did-What:

9703 (17.27) Q: Tottenham manager Juande Ramos has hinted he will allow ____ to leave
 9704 if the Bulgaria striker makes it clear he is unhappy. (Onishi et al., 2016)

9705 The query sentence may be selected either from the story itself, or from an external
 9706 summary. In either case, datasets can be created automatically by processing large
 9707 quantities existing documents. An additional constraint is that that missing element
 9708 from the cloze must appear in the main passage of text: for example, in Who-did-
 9709 What, the candidates include all entities mentioned in the main passage. In the
 9710 CNN/Daily Mail dataset, each entity name is replaced by a unique identifier, e.g.,
 9711 ENTITY37. This ensures that correct answers can only be obtained by accurately
 9712 reading the text, and not from external knowledge about the entities.

- 9713 • **Extractive** question answering, in which the answer is drawn from the original text.
 9714 In WikiQA, answers are sentences (Yang et al., 2015). In the Stanford Question An-
 9715 swering Dataset (SQuAD), answers are words or short phrases (Rajpurkar et al.,
 9716 2016):

9717 (17.28) In metereology, precipitation is any product of the condensation of atmo-
 9718 spheric water vapor that falls under gravity.
 9719 Q: What causes precipitation to fall? A: gravity

9720 In both WikiQA and SQuAD, the original texts are Wikipedia articles, and the ques-
 9721 tions are generated by crowdworkers.

9722 17.5.2.2 Methods

9723 A baseline method is to search the text for sentences or short passages that overlap with
 9724 both the query and the candidate answer (Richardson et al., 2013). In example (17.26), this
 9725 baseline would select the correct answer, since *James* appears in a sentence that includes
 9726 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,

$$\hat{c} = \underset{c}{\operatorname{argmax}} \mathbf{o} \cdot \mathbf{x}_c. \quad [17.29]$$

9727 This architecture can be trained end-to-end from a loss based on the log-likelihood of the
 9728 correct answer. A number of related architectures have been proposed (e.g., Hermann
 9729 et al., 2015; Kadlec et al., 2016; Dhingra et al., 2017; Cui et al., 2017), and these methods are
 9730 surveyed by Wang et al. (2017).

9731 Additional resources

9732 The field of information extraction is surveyed in course notes by Grishman (2012), and
 9733 more recently in a short survey paper (Grishman, 2015). Shen et al. (2015) survey the task
 9734 of entity linking, and Ji and Grishman (2011) survey work on knowledge base popula-
 9735 tion. This chapter’s discussion of non-propositional meaning was strongly influenced by
 9736 Morante and Sporleder (2012), who introduced a special issue of the journal *Computational
 9737 Linguistics* dedicated to recent work on modality and negation.

9738 Exercises

- 9739 1. Go to the Wikipedia page for your favorite movie. For each record in the info box
 9740 (e.g., *Screenplay by: Stanley Kubrick*), report whether there is a sentence in the ar-
 9741 ticle containing both the field and value (e.g., *The screenplay was written by Stanley
 9742 Kubrick*). If not, is there a sentence in the article containing just the value? (For
 9743 records with more than one value, just use the first value.)
- 9744 2. Building on your answer in the previous question, report the dependency path be-
 9745 tween the head words of the field and value for at least three records.
- 9746 3. Consider the following heuristic for entity linking:

- 9747 • Among all entities that have the same type as the mention (e.g., LOC, PER),
 9748 choose the one whose name has the lowest edit distance from the mention.
 9749 • If more than one entity has the right type and the lowest edit distance from the
 9750 mention, choose the most popular one.
 9751 • If no candidate entity has the right type, choose NIL.

Now suppose you have the following feature function:

$$f(y, \mathbf{x}) = [\text{edit-dist}(\text{name}(y), \mathbf{x}), \text{same-type}(y, \mathbf{x}), \text{popularity}(y), \delta(y = \text{NIL})]$$

9752 Design a set of ranking weights θ that match the heuristic. You may assume that
 9753 edit distance and popularity are always in the range [0, 100], and that the NIL entity
 9754 has values of zero for all features except $\delta(y = \text{NIL})$.

9755 4. Now consider another heuristic:

- 9756 • Among all candidate entities that have edit distance zero from the mention and
 9757 the right type, choose the most popular one.
 9758 • If no entity has edit distance zero from the mention, choose the one with the
 9759 right type that is most popular, regardless of edit distance.
 9760 • If no entity has the right type, choose NIL.

9761 Using the same features and assumptions from the previous problem, prove that
 9762 there is no set of weights that could implement this heuristic. Then show that the
 9763 heuristic can be implemented by adding a single feature. Your new feature should
 9764 consider only the edit distance.

9765 5. * Consider the following formulation for collective entity linking, which rewards
 9766 sets of entities that are all of the same type, where “types” can be elements of any
 9767 set:

$$\psi_c(\mathbf{y}) = \begin{cases} \alpha & \text{all entities in } \mathbf{y} \text{ have the same type} \\ \beta & \text{more than half of the entities in } \mathbf{y} \text{ have the same type} \\ 0 & \text{otherwise.} \end{cases} \quad [17.30]$$

9768 Show how to implement this model of collective entity linking in an **integer linear**
 9769 **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., $s_{i,y} = \theta \cdot \mathbf{f}(y, \mathbf{x}^{(i)}, \mathbf{c}^{(i)})$.

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.

6. Run `nltk.corpus.download('reuters')` to download the Reuters corpus in NLTK, and run from `nltk.corpus import reuters` to import it. The command `reuters.words()` returns an iterator over the tokens in the corpus.
- a) Apply the pattern *_____, such as ____* to this corpus, obtaining candidates for the IS-A relation, e.g. IS-A(ROMANIA, COUNTRY). What are three pairs that this method identifies correctly? What are three different pairs that it gets wrong?
 - b) Design a pattern for the PRESIDENT relation, e.g. PRESIDENT(PHILIPPINES, CORAZON AQUINO). In this case, you may want to augment your pattern matcher with the ability to match multiple token wildcards, perhaps using case information to detect proper names. Again, list three correct
 - c) Preprocess the Reuters data by running a named entity recognizer, replacing tokens with named entity spans when applicable. Apply your PRESIDENT matcher to this new data. Does the accuracy improve? Compare 20 randomly-selected pairs from this pattern and the one you designed in the previous part.
7. Represent the dependency path $\mathbf{x}^{(i)}$ as a sequence of words and dependency arcs of length M_i , ignoring the endpoints of the path. In example 1 of Table 17.2, the dependency path is,

$$\mathbf{x}^{(1)} = (\xleftarrow[\text{NSUBJ}]{} \text{traveled}, \xrightarrow[\text{OBL}]{}) \quad [17.32]$$

If $x_m^{(i)}$ is a word, then let $\text{pos}(x_m^{(i)})$ be its part-of-speech, using the tagset defined in chapter 8.

We can define the following kernel function over pairs of dependency paths (Bunescu and Mooney, 2005):

$$\kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \begin{cases} 0, & M_i \neq M_j \\ \prod_{m=1}^{M_i} c(x_m^{(i)}, x_m^{(j)}), & M_i = M_j \end{cases}$$

$$c(x_m^{(i)}, x_m^{(j)}) = \begin{cases} 2, & x_m^{(i)} = x_m^{(j)} \\ 1, & x_m^{(i)} \text{ and } x_m^{(j)} \text{ are words and } \text{pos}(x_m^{(i)}) = \text{pos}(x_m^{(j)}) \\ 0, & \text{otherwise.} \end{cases}$$

Using this kernel function, compute the kernel similarities of example 1 from Table 17.2 with the other five examples.

- 9793 8. Continuing from the previous problem, suppose that the instances have the follow-
 9794 ing labels:

$$y_2 = 1, y_3 = -1, y_4 = -1, y_5 = 1, y_6 = 1 \quad [17.33]$$

9795 Identify the conditions for α and b under which $\hat{y}_1 = 1$. Remember the constraint
 9796 that $\alpha_i \geq 0$ for all i .

9797 9. Using the `reuters` corpus in `nltk`, apply distant supervision to build a training
 9798 set for detecting the relation between nations and their capitals. Start with the fol-
 9799 lowing known relations: (JAPAN, TOKYO), (FRANCE, PARIS), (ITALY, ROME). How
 9800 many positive and negative examples are you able to extract?

9801 10. Identify the focus and scope of negation in the following examples:

- 9802 (17.29) a. As no better man advances to take this matter in hand, I hereupon offer
 9803 my own poor endeavors.
 9804 b. That's no way to convert sinners, cook!
 9805 c. And concerning all these, is not Possession the whole of the law?
 9806 d. And had Flask helped himself, the chances were Ahab had never so
 9807 much as noticed it.

9808 11. Consider the neural QA system described in § 17.5.2.2, but restrict the set of can-
 9809 didate answers to words in the passage, and set each candidate answer embed-
 9810 ding \mathbf{x} equal to the vector $\mathbf{h}_m^{(p)}$, representing token m in the passage, so that $\hat{m} =$
 9811 $\text{argmax}_m \mathbf{o} \cdot \mathbf{h}_m^{(p)}$. Suppose the system selects answer \hat{m} , but the correct answer is m^* .
 9812 Consider the gradient of the margin loss with respect to the attention:

- 9813 a) Prove that $\frac{\partial \ell}{\partial \alpha_{\hat{m}}} \geq \frac{\partial \ell}{\partial \alpha_{m^*}}$.

- 9814 b) Assuming that $\|\mathbf{h}_{\hat{m}}\| = \|\mathbf{h}_{m^*}\|$, prove that $\frac{\partial \ell}{\partial \alpha_{\hat{m}}} \geq 0$ and $\frac{\partial \ell}{\partial \alpha_{m^*}} \leq 0$. Explain in
9815 words what this means about how the attention is expected to change after a
9816 gradient-based update.

9817 Chapter 18

9818 Machine translation

9819 Machine translation (MT) is one of the “holy grail” problems in artificial intelligence,
9820 with the potential to transform society by facilitating communication between people
9821 anywhere in the world. As a result, MT has received significant attention and funding
9822 since the early 1950s. However, it has proved remarkably challenging, and while there
9823 has been substantial progress towards usable MT systems — especially for high-resource
9824 language pairs like English-French — we are still far from translation systems that match
9825 the nuance and depth of human translations.

9826 18.1 Machine translation as a task

9827 Machine translation can be formulated as an optimization problem:

$$\hat{\mathbf{w}}^{(t)} = \underset{\mathbf{w}^{(t)}}{\operatorname{argmax}} \Psi(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}), \quad [18.1]$$

9828 where $\mathbf{w}^{(s)}$ is a sentence in a **source** language, $\mathbf{w}^{(t)}$ is a sentence in the **target language**,
9829 and Ψ is a scoring function. As usual, this formalism requires two components: a decod-
9830 ing algorithm for computing $\hat{\mathbf{w}}^{(t)}$, and a learning algorithm for estimating the parameters
9831 of the scoring function Ψ .

9832 Decoding is difficult for machine translation because of the huge space of possible
9833 translations. We have faced large label spaces before: for example, in sequence labeling,
9834 the set of possible label sequences is exponential in the length of the input. In these cases,
9835 it was possible to search the space quickly by introducing locality assumptions: for ex-
9836 ample, that each tag depends only on its predecessor, or that each production depends
9837 only on its parent. In machine translation, no such locality assumptions seem possible:
9838 human translators reword, reorder, and rearrange words; they replace single words with
9839 multi-word phrases, and vice versa. This flexibility means that in even relatively simple

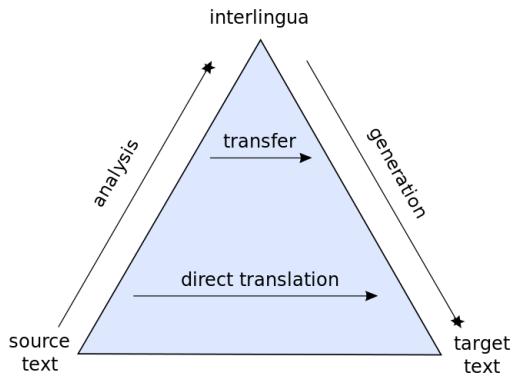


Figure 18.1: The Vauquois Pyramid http://commons.wikimedia.org/wiki/File:Direct_translation_and_transfer_translation_pyramind.svg

9840 translation models, decoding is NP-hard (Knight, 1999). Approaches for dealing with this
 9841 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.,

$$\begin{aligned} w^{(s)} &= A \text{ Vinay le gusta las manzanas.} \\ w^{(t)} &= \text{Vinay likes apples.} \end{aligned}$$

9842 A useful feature function would note the translation pairs (*gusta, likes*), (*manzanas, apples*),
 9843 and even (*Vinay, Vinay*). But this word-to-word **alignment** is not given in the data. One
 9844 solution is to treat this alignment as a **latent variable**; this is the approach taken by clas-
 9845 sical **statistical machine translation** (SMT) systems, described in § 18.2. Another solution
 9846 is to model the relationship between $w^{(t)}$ and $w^{(s)}$ through a more complex and expres-
 9847 sive function; this is the approach taken by **neural machine translation** (NMT) systems,
 9848 described in § 18.3.

9849 The **Vauquois Pyramid** is a theory of how translation should be done. At the lowest
 9850 level, the translation system operates on individual words, but the horizontal distance
 9851 at this level is large, because languages express ideas differently. If we can move up the
 9852 triangle to syntactic structure, the distance for translation is reduced; we then need only
 9853 produce target-language text from the syntactic representation, which can be as simple
 9854 as reading off a tree. Further up the triangle lies semantics; translating between semantic
 9855 representations should be easier still, but mapping between semantics and surface text is a
 9856 difficult, unsolved problem. At the top of the triangle is **interlingua**, a semantic represen-
 9857 tation that is so generic that it is identical across all human languages. Philosophers de-
 9858 bate whether such a thing as interlingua is really possible (e.g., Derrida, 1985). While the

	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*.

first-order logic representations discussed in chapter 12 might be thought to be language independent, they are built on an inventory of predicates that are suspiciously similar to 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 becomes 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.

The BLEU score is then based on the average, $\exp \frac{1}{N} \sum_{n=1}^N \log p_n$. Two modifications of Equation 18.2 are necessary: (1) to avoid computing $\log 0$, all precisions are smoothed

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.

to ensure that they are positive; (2) each n -gram in the reference can be used at most once, so that *to to to to to* does not achieve $p_1 = 1$ against the reference *to be or not to be*. Furthermore, precision-based metrics are biased in favor of short translations, which can achieve high scores by minimizing the denominator in [18.2]. To avoid this issue, a **brevity penalty** is applied to translations that are shorter than the reference. This penalty is indicated as “BP” in Figure 18.2.

Automated metrics like BLEU have been validated by correlation with human judgments of translation quality. Nonetheless, it is not difficult to construct examples in which the BLEU score is high, yet the translation is disfluent or carries a completely different meaning from the original. To give just one example, consider the problem of translating pronouns. Because pronouns refer to specific entities, a single incorrect pronoun can obliterate the semantics of the original sentence. Existing state-of-the-art systems generally do not attempt the reasoning necessary to correctly resolve pronominal anaphora (Hartmeier, 2012). Despite the importance of pronouns for semantics, they have a marginal impact on BLEU, which may help to explain why existing systems do not make a greater effort to translate them correctly.

Fairness and bias The problem of pronoun translation intersects with issues of fairness and bias. In many languages, such as Turkish, the third person singular pronoun is gender neutral. Today’s state-of-the-art systems produce the following Turkish-English translations (Caliskan et al., 2017):

- 9902 (18.1) *O bir doktor.*
He is a doctor.
- 9903 (18.2) *O bir hemşire.*
She is a nurse.

The same problem arises for other professions that have stereotypical genders, such as engineers, soldiers, and teachers, and for other languages that have gender-neutral pro-

9906 nouns. This bias was not directly programmed into the translation model; it arises from
9907 statistical tendencies in existing datasets. This highlights a general problem with data-
9908 driven approaches, which can perpetuate biases that negatively impact disadvantaged
9909 groups. Worse, machine learning can *amplify* biases in data (Bolukbasi et al., 2016): if a
9910 dataset has even a slight tendency towards men as doctors, the resulting translation model
9911 may produce translations in which doctors are always *he*, and nurses are always *she*.

9912 **Other metrics** A range of other automated metrics have been proposed for machine
9913 translation. One potential weakness of BLEU is that it only measures precision; METEOR
9914 is a weighted *F*-MEASURE, which is a combination of recall and precision (see § 4.4.1).
9915 **Translation Error Rate (TER)** computes the string **edit distance** (see § 9.1.4.1) between the
9916 reference and the hypothesis (Snover et al., 2006). For language pairs like English and
9917 Japanese, there are substantial differences in word order, and word order errors are not
9918 sufficiently captured by *n*-gram based metrics. The **RIBES** metric applies rank correla-
9919 tion to measure the similarity in word order between the system and reference transla-
9920 tions (Isozaki et al., 2010).

9921 18.1.2 Data

9922 Data-driven approaches to machine translation rely primarily on **parallel corpora**: sentence-
9923 level translations. Early work focused on government records, in which fine-grained offi-
9924 cial translations are often required. For example, the IBM translation systems were based
9925 on the proceedings of the Canadian Parliament, called **Hansards**, which are recorded in
9926 English and French (Brown et al., 1990). The growth of the European Union led to the
9927 development of the **EuroParl corpus**, which spans 21 European languages (Koehn, 2005).
9928 While these datasets helped to launch the field of machine translation, they are restricted
9929 to narrow domains and a formal speaking style, limiting their applicability to other types
9930 of text. As more resources are committed to machine translation, new translation datasets
9931 have been commissioned. This has broadened the scope of available data to news,² movie
9932 subtitles,³ social media (Ling et al., 2013), dialogues (Fordyce, 2007), TED talks (Paul et al.,
9933 2010), and scientific research articles (Nakazawa et al., 2016).

9934 Despite this growing set of resources, the main bottleneck in machine translation data
9935 is the need for parallel corpora that are aligned at the sentence level. Many languages have
9936 sizable parallel corpora with some high-resource language, but not with each other. The
9937 high-resource language can then be used as a “pivot” or “bridge” (Boitet, 1988; Utiyama
9938 and Isahara, 2007): for example, De Gispert and Marino (2006) use Spanish as a bridge for
9939 translation between Catalan and English. For most of the 6000 languages spoken today,

²https://catalog.ldc.upenn.edu/LDC2010T10_translation-task.html <http://www.statmt.org/wmt15/>

³<http://opus.nlpl.eu/>

9940 the only source of translation data remains the Judeo-Christian Bible (Resnik et al., 1999).
 9941 While relatively small, at less than a million tokens, the Bible has been translated into
 9942 more than 2000 languages, far outpacing any other corpus. Some research has explored
 9943 the possibility of automatically identifying parallel sentence pairs from unaligned parallel
 9944 texts, such as web pages and Wikipedia articles (Kilgarriff and Grefenstette, 2003; Resnik
 9945 and Smith, 2003; Adafre and De Rijke, 2006). Another approach is to create large parallel
 9946 corpora through crowdsourcing (Zaidan and Callison-Burch, 2011).

9947 18.2 Statistical machine translation

9948 The previous section introduced adequacy and fluency as the two main criteria for ma-
 9949 chine translation. A natural modeling approach is to represent them with separate scores,

$$\Psi(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \Psi_A(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) + \Psi_F(\mathbf{w}^{(t)}). \quad [18.3]$$

9950 The fluency score Ψ_F need not even consider the source sentence; it only judges $\mathbf{w}^{(t)}$ on
 9951 whether it is fluent in the target language. This decomposition is advantageous because
 9952 it makes it possible to estimate the two scoring functions on separate data. While the
 9953 adequacy model must be estimated from aligned sentences — which are relatively expen-
 9954 sive and rare — the fluency model can be estimated from monolingual text in the target
 9955 language. Large monolingual corpora are now available in many languages, thanks to
 9956 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:

$$\Psi_A(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) \triangleq \log p_{S|T}(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}) \quad [18.4]$$

$$\Psi_F(\mathbf{w}^{(t)}) \triangleq \log p_T(\mathbf{w}^{(t)}) \quad [18.5]$$

$$\Psi(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \log p_{S|T}(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}) + \log p_T(\mathbf{w}^{(t)}) = \log p_{S,T}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}). \quad [18.6]$$

9957 By setting the scoring functions equal to the logarithms of the prior and likelihood, their
 9958 sum is equal to $\log p_{S,T}$, which is the logarithm of the joint probability of the source and
 9959 target. The sentence $\hat{\mathbf{w}}^{(t)}$ that maximizes this joint probability is also the maximizer of the
 9960 conditional probability $p_{T|S}$, making it the most likely target language sentence, condi-
 9961 tioned on the source.

9962 The noisy channel model can be justified by a generative story. The target text is orig-
 9963 inally generated from a probability model p_T . It is then encoded in a “noisy channel”
 9964 $p_{S|T}$, which converts it to a string in the source language. In decoding, we apply Bayes’
 9965 rule to recover the string $\mathbf{w}^{(t)}$ that is maximally likely under the conditional probability
 9966 $p_{T|S}$. Under this interpretation, the target probability p_T is just a language model, and
 9967 can be estimated using any of the techniques from chapter 6. The only remaining learning
 9968 problem is to estimate the translation model $p_{S|T}$.

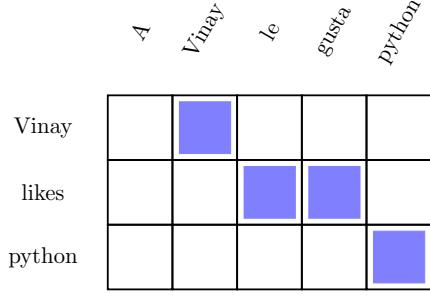


Figure 18.3: An example word-to-word alignment

18.2.1 Statistical translation modeling

The simplest decomposition of the translation model is word-to-word: each word in the source should be aligned to a word in the translation. This approach presupposes an **alignment** $\mathcal{A}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)})$, which contains a list of pairs of source and target tokens. For example, given $\mathbf{w}^{(s)} = A\ Vinay\ le\ gusta\ Python$ and $\mathbf{w}^{(t)} = Vinay\ likes\ Python$, one possible word-to-word alignment is,

$$\mathcal{A}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \{(A, \emptyset), (Vinay, Vinay), (le, likes), (gusta, likes), (Python, Python)\}. \quad [18.7]$$

This alignment is shown in Figure 18.3. Another, less promising, alignment is:

$$\mathcal{A}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \{(A, Vinay), (Vinay, likes), (le, Python), (gusta, \emptyset), (Python, \emptyset)\}. \quad [18.8]$$

Each alignment contains exactly one tuple for each word in the *source*, which serves to explain how the source word could be translated from the target, as required by the translation probability $p_{S|T}$. If no appropriate word in the target can be identified for a source word, it is aligned to \emptyset — as is the case for the Spanish function word *a* in the example, which glosses to the English word *to*. Words in the target can align with multiple words 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,

$$p(\mathbf{w}^{(s)}, \mathcal{A} | \mathbf{w}^{(t)}) = \prod_{m=1}^{M^{(s)}} p(w_m^{(s)}, a_m | w_{a_m}^{(t)}, m, M^{(s)}, M^{(t)}) \quad [18.9]$$

$$= \prod_{m=1}^{M^{(s)}} p(a_m | m, M^{(s)}, M^{(t)}) \times p(w_m^{(s)} | w_{a_m}^{(t)}). \quad [18.10]$$

This probability model makes two key assumptions:

- 9983 • The alignment probability factors across tokens,

$$p(\mathcal{A} \mid \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \prod_{m=1}^{M^{(s)}} p(a_m \mid m, M^{(s)}, M^{(t)}). \quad [18.11]$$

9984 This means that each alignment decision is independent of the others, and depends
 9985 only on the index m , and the sentence lengths $M^{(s)}$ and $M^{(t)}$.

- 9986 • The translation probability also factors across tokens,

$$p(\mathbf{w}^{(s)} \mid \mathbf{w}^{(t)}, \mathcal{A}) = \prod_{m=1}^{M^{(s)}} p(w_m^{(s)} \mid w_{a_m}^{(t)}), \quad [18.12]$$

9987 so that each word in $\mathbf{w}^{(s)}$ depends only on its aligned word in $\mathbf{w}^{(t)}$. This means that
 9988 translation is word-to-word, ignoring context. The hope is that the target language
 9989 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,

$$p(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \sum_{\mathcal{A}} p(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}, \mathcal{A}) \quad [18.13]$$

$$= p(\mathbf{w}^{(t)}) \sum_{\mathcal{A}} p(\mathcal{A}) \times p(\mathbf{w}^{(s)} \mid \mathbf{w}^{(t)}, \mathcal{A}) \quad [18.14]$$

$$\geq p(\mathbf{w}^{(t)}) \max_{\mathcal{A}} p(\mathcal{A}) \times p(\mathbf{w}^{(s)} \mid \mathbf{w}^{(t)}, \mathcal{A}). \quad [18.15]$$

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:

$$p(a_m \mid m, M^{(s)}, M^{(t)}) = \frac{1}{M^{(t)}}. \quad [18.16]$$

9990 In this model, every alignment is equally likely. This is almost surely wrong, but it re-
 9991 sults in a convex learning objective, yielding a good initialization for the more complex
 9992 alignment models (Brown et al., 1993; Koehn, 2009).

9993 18.2.2 Estimation

9994 Let us define the parameter $\theta_{u \rightarrow v}$ as the probability of translating target word u to source
 9995 word v . If word-to-word alignments were annotated, these probabilities could be com-
 9996 puted from relative frequencies,

$$\hat{\theta}_{u \rightarrow v} = \frac{\text{count}(u, v)}{\text{count}(u)}, \quad [18.17]$$

9997 where $\text{count}(u, v)$ is the count of instances in which word v was aligned to word u in
 9998 the training set, and $\text{count}(u)$ is the total count of the target word u . The smoothing
 9999 techniques mentioned in chapter 6 can help to reduce the variance of these probability
 10000 estimates.

10001 Conversely, if we had an accurate translation model, we could estimate the likelihood
 10002 of each alignment decision,

$$q_m(a_m \mid \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) \propto p(a_m \mid m, M^{(s)}, M^{(t)}) \times p(w_m^{(s)} \mid w_{a_m}^{(t)}), \quad [18.18]$$

where $q_m(a_m \mid \mathbf{w}^{(s)}, \mathbf{w}^{(t)})$ 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*,

$$\hat{\theta}_{u \rightarrow v} = \frac{E_q [\text{count}(u, v)]}{\text{count}(u)} \quad [18.19]$$

$$E_q [\text{count}(u, v)] = \sum_m q_m(a_m \mid \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) \times \delta(w_m^{(s)} = v) \times \delta(w_{a_m}^{(t)} = u). \quad [18.20]$$

10003 The **expectation-maximization** (EM) algorithm proceeds by iteratively updating q_m
 10004 and $\hat{\Theta}$. The algorithm is described in general form in chapter 5. For statistical machine
 10005 translation, the steps of the algorithm are:

- 10006 1. **E-step:** Update beliefs about word alignment using Equation 18.18.
- 10007 2. **M-step:** Update the translation model using Equations 18.19 and 18.20.

10008 As discussed in chapter 5, the expectation maximization algorithm is guaranteed to con-
 10009 verge, but not to a global optimum. However, for IBM Model 1, it can be shown that EM
 10010 optimizes a convex objective, and global optimality is guaranteed. For this reason, IBM
 10011 Model 1 is often used as an initialization for more complex alignment models. For more
 10012 detail, see Koehn (2009).

10013 18.2.3 Phrase-based translation

10014 Real translations are not word-to-word substitutions. One reason is that many multiword
 10015 expressions are not translated literally, as shown in this example from French:

- 10016 (18.3) *Nous allons prendre un verre*
 We will take a glass
 10017 We'll have a drink

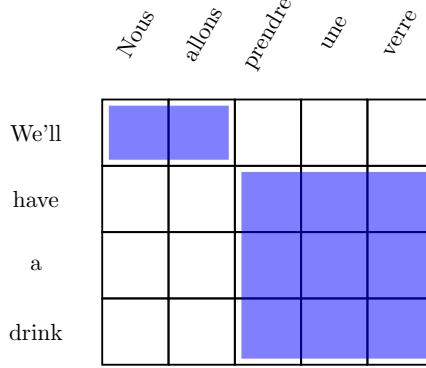


Figure 18.4: A phrase-based alignment between French and English, corresponding to example (18.3)

10018 The line *we will take a glass* is the word-for-word gloss of the French sentence; the transla-
 10019 tion *we'll have a drink* is shown on the third line. Such examples are difficult for word-to-
 10020 word translation models, since they require translating *prendre* to *have* and *verre* to *drink*.
 10021 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

$$p(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}, \mathcal{A}) = \prod_{((i, j), (k, \ell)) \in \mathcal{A}} p_{w^{(s)}|w^{(t)}}(\{w_{i+1}^{(s)}, w_{i+2}^{(s)}, \dots, w_j^{(s)}\} | \{w_{k+1}^{(t)}, w_{k+2}^{(t)}, \dots, w_\ell^{(t)}\}). \quad [18.21]$$

10022 The phrase alignment $((i, j), (k, \ell))$ indicates that the span $\mathbf{w}_{i+1:j}^{(s)}$ is the translation of the
 10023 span $\mathbf{w}_{k+1:\ell}^{(t)}$. An example phrasal alignment is shown in Figure 18.4. Note that the align-
 10024 ment set \mathcal{A} is required to cover all of the tokens in the source, just as in word-based trans-
 10025 lation. The probability model $p_{w^{(s)}|w^{(t)}}$ must now include translations for all phrase pairs,
 10026 which can be learned from expectation-maximization just as in word-based statistical ma-
 10027 chine translation.

10028 **18.2.4 *Syntax-based translation**

10029 The Vauquois Pyramid (Figure 18.1) suggests that translation might be easier if we take a
 10030 higher-level view. One possibility is to incorporate the syntactic structure of the source,
 10031 the target, or both. This is particularly promising for language pairs that consistent syn-
 10032 tactic differences. For example, English adjectives almost always precede the nouns that
 10033 they modify, while in Romance languages such as French and Spanish, the adjective often
 10034 follows the noun: thus, *angry fish* would translate to *pez (fish) enojado (angry)* in Spanish.
 10035 In word-to-word translation, these reorderings cause the alignment model to be overly
 10036 permissive. It is not that the order of *any* pair of English words can be reversed when
 10037 translating into Spanish, but only adjectives and nouns within a noun phrase. Similar
 10038 issues arise when translating between verb-final languages such as Japanese (in which
 10039 verbs usually follow the subject and object), verb-initial languages like Tagalog and clas-
 10040 sical Arabic, and verb-medial languages such as English.

10041 An elegant solution is to link parsing and translation in a **synchronous context-free**
 10042 **grammar** (SCFG; Chiang, 2007).⁴ An SCFG is a set of productions of the form $X \rightarrow (\alpha, \beta, \sim)$,
 10043 where X is a non-terminal, α and β are sequences of terminals or non-terminals, and \sim
 10044 is a one-to-one alignment of items in α with items in β . To handle the English-Spanish
 10045 adjective-noun ordering, an SCFG would include productions such as,

$$\text{NP} \rightarrow (\text{DET}_1 \text{NN}_2 \text{JJ}_3, \quad \text{DET}_1 \text{JJ}_3 \text{NN}_2), \quad [18.22]$$

10046 with subscripts indicating the alignment between the Spanish (left) and English (right)
 10047 parts of the right-hand side. Terminal productions yield translation pairs,

$$\text{JJ} \rightarrow (\text{enojado}_1, \text{angry}_1). \quad [18.23]$$

10048 A synchronous derivation begins with the start symbol S , and derives a pair of sequences
 10049 of terminal symbols.

10050 Given an SCFG in which each production yields at most two symbols in each language
 10051 (Chomsky Normal Form; see § 9.2.1.2), a sentence can be parsed using only the CKY
 10052 algorithm (chapter 10). The resulting derivation also includes productions in the other
 10053 language, all the way down to the surface form. Therefore, SCFGs make translation very
 10054 similar to parsing. In a weighted SCFG, the log probability $\log p_{S|T}$ can be computed from
 10055 the sum of the log-probabilities of the productions. However, combining SCFGs with a
 10056 target language model is computationally expensive, necessitating approximate search
 10057 algorithms (Huang and Chiang, 2007).

10058 Synchronous context-free grammars are an example of **tree-to-tree translation**, be-
 10059 cause they model the syntactic structure of both the target and source language. In **string-**
 10060 **to-tree translation**, string elements are translated into constituent tree fragments, which

⁴Key earlier work includes syntax-driven transduction (Lewis II and Stearns, 1968) and stochastic inver-
 sion transduction grammars (Wu, 1997).

10061 are then assembled into a translation (Yamada and Knight, 2001; Galley et al., 2004); in
 10062 **tree-to-string translation**, the source side is parsed, and then transformed into a string on
 10063 the target side (Liu et al., 2006). A key question for syntax-based translation is the extent
 10064 to which we phrasal constituents align across translations (Fox, 2002), because this gov-
 10065 erns the extent to which we can rely on monolingual parsers and treebanks. For more on
 10066 syntax-based machine translation, see the monograph by Williams et al. (2016).

10067 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.

$$\mathbf{z} = \text{ENCODE}(\mathbf{w}^{(s)}) \quad [18.24]$$

$$\mathbf{w}^{(t)} \mid \mathbf{w}^{(s)} \sim \text{DECODE}(\mathbf{z}), \quad [18.25]$$

10068 where the second line means that the function $\text{DECODE}(\mathbf{z})$ defines the conditional proba-
 10069 bility $p(\mathbf{w}^{(t)} \mid \mathbf{w}^{(s)})$.

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,

$$\log p(\mathbf{w}^{(t)} \mid \mathbf{w}^{(s)}) = \sum_{m=1}^{M^{(t)}} p(w_m^{(t)} \mid \mathbf{w}_{1:m-1}^{(t)}, \mathbf{z}) \quad [18.26]$$

$$p(w_m^{(t)} \mid \mathbf{w}_{1:m-1}^{(t)}, \mathbf{w}^{(s)}) \propto \exp\left(\boldsymbol{\beta}_{w_m^{(t)}} \cdot \mathbf{h}_{m-1}^{(t)}\right) \quad [18.27]$$

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 \mathbf{z} . The second line is equivalent to writing,

$$w_m^{(t)} \mid \mathbf{w}_{1:m-1}^{(t)}, \mathbf{w}^{(s)} \sim \text{SoftMax}\left(\boldsymbol{\beta} \cdot \mathbf{h}_{m-1}^{(t)}\right), \quad [18.28]$$

10070 where $\boldsymbol{\beta} \in \mathbb{R}^{(V^{(t)} \times K)}$ is the matrix of output word vectors for the $V^{(t)}$ words in the target
 10071 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**

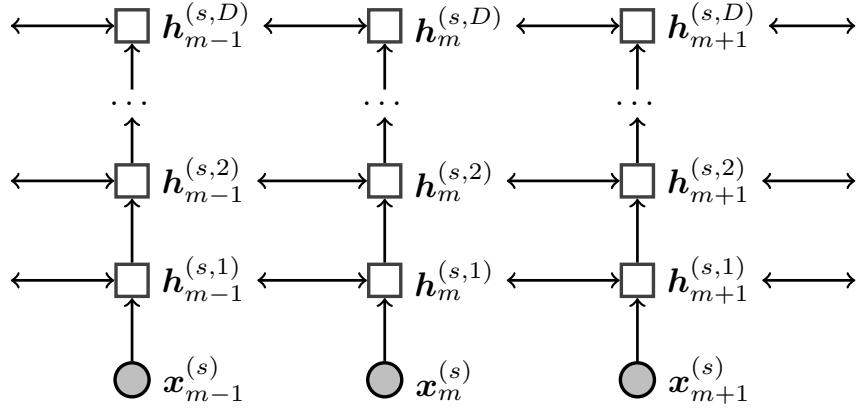


Figure 18.5: A deep bidirectional LSTM encoder

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:

$$\mathbf{h}_0^{(t)} = \mathbf{z} \quad [18.31]$$

$$\mathbf{h}_m^{(t)} = \text{LSTM}(\mathbf{x}_m^{(t)}, \mathbf{h}_{m-1}^{(t)}), \quad [18.32]$$

10072 where $\mathbf{x}_m^{(t)}$ is the embedding of the target language word $w_m^{(t)}$.

10073 Sequence-to-sequence translation is nothing more than wiring together two LSTMs:
 10074 one to read the source, and another to generate the target. To make the model work well,
 10075 some additional tweaks are needed:

- 10076 • Most notably, the model works much better if the source sentence is reversed, reading
 10077 from the end of the sentence back to the beginning. In this way, the words at the
 10078 beginning of the source have the greatest impact on the encoding \mathbf{z} , and therefore
 10079 impact the words at the beginning of the target sentence. Later work on more advanced
 10080 encoding models, such as **neural attention** (see § 18.3.1), has eliminated the
 10081 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 $i + 1$:

$$\mathbf{h}_m^{(s,1)} = \text{LSTM}(\mathbf{x}_m^{(s)}, \mathbf{h}_{m-1}^{(s)}) \quad [18.33]$$

$$\mathbf{h}_m^{(s,i+1)} = \text{LSTM}(\mathbf{h}_m^{(s,i)}, \mathbf{h}_{m-1}^{(s,i+1)}), \quad \forall i \geq 1. \quad [18.34]$$

10082 The original work on sequence-to-sequence translation used four layers; in 2016,
 10083 Google’s commercial machine translation system used eight layers (Wu et al., 2016).⁵

- 10084 • Significant improvements can be obtained by creating an **ensemble** of translation
 10085 models, each trained from a different random initialization. For an ensemble of size
 10086 N , the per-token decoding probability is set equal to,

$$p(w^{(t)} | z, \mathbf{w}_{1:m-1}^{(t)}) = \frac{1}{N} \sum_{i=1}^N p_i(w^{(t)} | z, \mathbf{w}_{1:m-1}^{(t)}), \quad [18.35]$$

10087 where p_i is the decoding probability for model i . Each translation model in the
 10088 ensemble includes its own encoder and decoder networks.

- 10089 • The original sequence-to-sequence model used a fairly standard training setup: stochastic
 10090 gradient descent with an exponentially decreasing learning rate after the first five
 10091 epochs; mini-batches of 128 sentences, chosen to have similar length so that each
 10092 sentence on the batch will take roughly the same amount of time to process; gradient
 10093 clipping (see § 3.3.4) to ensure that the norm of the gradient never exceeds some
 10094 predefined value.

10095 18.3.1 Neural attention

10096 The sequence-to-sequence model discussed in the previous section was a radical departure
 10097 from statistical machine translation, in which each word or phrase in the target lan-
 10098 guage is conditioned on a single word or phrase in the source language. Both approaches
 10099 have advantages. Statistical translation leverages the idea of compositionality — transla-
 10100 tions of large units should be based on the translations of their component parts — and
 10101 this seems crucial if we are to scale translation to longer units of text. But the translation
 10102 of each word or phrase often depends on the larger context, and encoder-decoder models
 10103 capture this context at the sentence level.

10104 Is it possible for translation to be both contextualized and compositional? One ap-
 10105 proach is to augment neural translation with an **attention mechanism**. The idea of neural
 10106 attention was described in § 17.5, but its application to translation bears further discus-
 10107 sion. In general, attention can be thought of as using a query to select from a memory
 10108 of key-value pairs. However, the query, keys, and values are all vectors, and the entire

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.

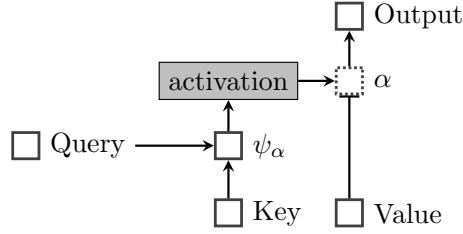


Figure 18.6: A general view of neural attention. The dotted box indicates that each $\alpha_{m \rightarrow n}$ can be viewed as a **gate** on value n .

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 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 \mathbf{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):

$$\tilde{h}_m^{(t)} = \tanh(\Theta_c[h_m^{(t)}; c_m]) \quad [18.40]$$

$$p(w_{m+1}^{(t)} | w_{1:m}^{(t)}, w^{(s)}) \propto \exp\left(\beta_{w_{m+1}^{(t)}} \cdot \tilde{h}_m^{(t)}\right). \quad [18.41]$$

10129 Here the decoder state $h_m^{(t)}$ is concatenated with the context vector, forming the input
 10130 to compute a final output vector $\tilde{h}_m^{(t)}$. The context vector can be incorporated into the
 10131 decoder recurrence in a similar manner (Bahdanau et al., 2014).

10132 **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:

$$z_m^{(i)} = \sum_{n=1}^{M^{(s)}} \alpha_{m \rightarrow n}^{(i)} (\Theta_v h_n^{(i-1)}) \quad [18.42]$$

$$h_m^{(i)} = \Theta_2 \text{ReLU}\left(\Theta_1 z_m^{(i)} + b_1\right) + b_2. \quad [18.43]$$

10133 For each token m at level i , we compute self-attention over the entire source sentence:
 10134 the keys, values, and queries are all projections of the vector $h^{(i-1)}$. The attention scores
 10135 $\alpha_{m \rightarrow n}^{(i)}$ are computed using a scaled form of softmax attention,

$$\alpha_{m \rightarrow n} \propto \exp(\psi_\alpha(m, n)/M), \quad [18.44]$$

10136 where M is the length of the input. This encourages the attention to be more evenly
 10137 dispersed across the input. Self-attention is applied across multiple “heads”, each using
 10138 different projections of $\mathbf{h}^{(i-1)}$ to form the keys, values, and queries.

The output of the self-attentional layer is the representation $\mathbf{z}_m^{(i)}$, which is then passed through a two-layer feed-forward network, yielding the input to the next layer, $\mathbf{h}^{(i)}$. To ensure that information about word order in the source is integrated into the model, the encoder includes **positional encodings** of the index of each word in the source. These encodings are vectors for each position $m \in \{1, 2, \dots, M\}$. The transformer sets these encodings equal to a set of sinusoidal functions of m ,

$$e_{2i-1}(m) = \sin(m/(10000^{\frac{2i}{K_e}})) \quad [18.45]$$

$$e_{2i}(m) = \cos(m/(10000^{\frac{2i}{K_e}})), \quad \forall i \in \{1, 2, \dots, K_e/2\} \quad [18.46]$$

10139 where $e_{2i}(m)$ is the value at position $2i$ of the encoding for position m . As we progress
 10140 through the dimensions of the encoding, we encounter sinusoidal functions of progres-
 10141 sively wider bandwidth. This enables the model to learn to attend by relative positions of
 10142 words. The positional encodings are concatenated with the word embeddings \mathbf{x}_m at the
 10143 base layer of the model.⁶

10144 Convolutional neural networks (see § 3.4) have also been applied as encoders in neu-
 10145 ral machine translation. For each word $w_m^{(s)}$, a convolutional network computes a rep-
 10146 resentation $\mathbf{h}_m^{(s)}$ from the embeddings of the word and its neighbors. This procedure is
 10147 applied several times, creating a deep convolutional network. The recurrent decoder then
 10148 computes a set of attention weights over these convolutional representations, using the
 10149 decoder’s hidden state $\mathbf{h}^{(t)}$ as the queries. This attention vector is used to compute a
 10150 weighted average over the outputs of *another* convolutional neural network of the source,
 10151 yielding an averaged representation \mathbf{c}_m , which is then fed into the decoder. As with the
 10152 transformer, speed is the main advantage over recurrent encoding models; another sim-
 10153 ilarity is that word order information is approximated through the use of positional en-
 10154 codings.⁷

10155 18.3.3 Out-of-vocabulary words

10156 Thus far, we have treated translation as a problem at the level of words or phrases. For
 10157 words that do not appear in the training data, all such models will struggle. There are
 10158 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.

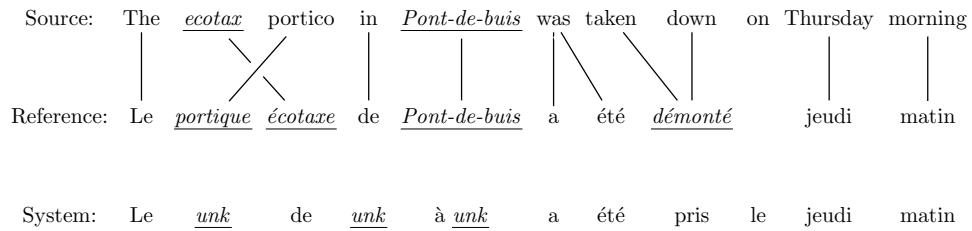


Figure 18.7: Translation with *unknown words*. The system outputs *unk* to indicate words that are outside its vocabulary. Figure adapted from Luong et al. (2015).

- 10159 • New proper nouns, such as family names or organizations, are constantly arising —
10160 particularly in the news domain. The same is true, to a lesser extent, for technical
10161 terminology. This issue is shown in Figure 18.7.
 - 10162 • In many languages, words have complex internal structure, known as **morphology**.
10163 An example is German, which uses compounding to form nouns like *Abwasserbe-*
10164 *handlungsanlage* (*sewage water treatment plant*; example from Sennrich et al. (2016)).
10165 While compounds could in principle be addressed by better tokenization (see § 8.4),
10166 other morphological processes involve more complex transformations of subword
10167 units.
- 10168 Names and technical terms can be handled in a postprocessing step: after first identi-
10169 fying alignments between unknown words in the source and target, we can look up each
10170 aligned source word in a dictionary, and choose a replacement (Luong et al., 2015). If the
10171 word does not appear in the dictionary, it is likely to be a proper noun, and can be copied
10172 directly from the source to the target. This approach can also be integrated directly into
10173 the translation model, rather than applying it as a postprocessing step (Jean et al., 2015).
- 10174 Words with complex internal structure can be handled by translating subword units
10175 rather than entire words. A popular technique for identifying subword units is **byte-pair**
10176 **encoding** (BPE; Gage, 1994; Sennrich et al., 2016). The initial vocabulary is defined as the
10177 set of characters used in the text. The most common character bigram is then merged into
10178 a new symbol, the vocabulary is updated, and the merging operation is applied again. For
10179 example, given the dictionary {fish, fished, want, wanted, bike, biked}, we would first form
10180 the subword unit ed, since this character bigram appears in three of the six words. Next,
10181 there are several bigrams that each appear in a pair of words: fi, is, sh, wa, an, etc. These can
10182 be merged in any order. By iterating this process, we eventually reach the segmentation,
10183 {fish, fish+ed, want, want+ed, bik+e, bik+ed}. At this point, there are no bigrams that appear
10184 more than once. In real data, merging is performed until the number of subword units
10185 reaches some predefined threshold, such as 10^4 .

10186 Each subword unit is treated as a token for translation, in both the encoder (source
 10187 side) and decoder (target side). BPE can be applied jointly to the union of the source and
 10188 target vocabularies, identifying subword units that appear in both languages. For lan-
 10189 guages that have different scripts, such as English and Russian, **transliteration** between
 10190 the scripts should be applied first.⁸

10191 **18.4 Decoding**

Given a trained translation model, the decoding task is:

$$\hat{\mathbf{w}}^{(t)} = \underset{\mathbf{w} \in \mathcal{V}^*}{\operatorname{argmax}} \Psi(\mathbf{w}, \mathbf{w}^{(s)}), \quad [18.47]$$

10192 where $\mathbf{w}^{(t)}$ is a sequence of tokens from the target vocabulary \mathcal{V} . It is not possible to
 10193 efficiently obtain exact solutions to the decoding problem, for even minimally effective
 10194 models in either statistical or neural machine translation. Today's state-of-the-art transla-
 10195 tion systems use **beam search** (see § 11.3.1.4), which is an incremental decoding algorithm
 10196 that maintains a small constant number of competitive hypotheses. Such greedy approxi-
 10197 mations are reasonably effective in practice, and this may be in part because the decoding
 10198 objective is only loosely correlated with measures of translation quality, so that exact op-
 10199 timization of [18.47] may not greatly improve the resulting translations.

Decoding in neural machine translation is simpler than in phrase-based statistical ma-
 chine translation.⁹ The scoring function Ψ is defined,

$$\Psi(\mathbf{w}^{(t)}, \mathbf{w}^{(s)}) = \sum_{m=1}^{M^{(t)}} \psi(w_m^{(t)}; \mathbf{w}_{1:m-1}^{(t)}, \mathbf{z}) \quad [18.48]$$

$$\psi(w^{(t)}; \mathbf{w}_{1:m-1}^{(t)}, \mathbf{z}) = \beta_{w_m^{(t)}} \cdot \mathbf{h}_m^{(t)} - \log \sum_{w \in \mathcal{V}} \exp(\beta_w \cdot \mathbf{h}_m^{(t)}), \quad [18.49]$$

10200 where \mathbf{z} is the encoding of the source sentence $\mathbf{w}^{(s)}$, and $\mathbf{h}_m^{(t)}$ is a function of the encoding
 10201 \mathbf{z} and the decoding history $\mathbf{w}_{1:m-1}^{(t)}$. This formulation subsumes the attentional translation
 10202 model, where \mathbf{z} is a matrix encoding of the source.

Now consider the incremental decoding algorithm,

$$\hat{w}_m^{(t)} = \underset{w \in \mathcal{V}}{\operatorname{argmax}} \psi(w; \hat{\mathbf{w}}_{1:m-1}^{(t)}, \mathbf{z}), \quad m = 1, 2, \dots \quad [18.50]$$

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).

10203 This algorithm selects the best target language word at position m , assuming that it has
 10204 already generated the sequence $\hat{w}_{1:m-1}^{(t)}$. (Termination can be handled by augmenting
 10205 the vocabulary \mathcal{V} with a special end-of-sequence token, ■.) The incremental algorithm
 10206 is likely to produce a suboptimal solution to the optimization problem defined in Equa-
 10207 tion 18.47, because selecting the highest-scoring word at position m can set the decoder
 10208 on a “garden path,” in which there are no good choices at some later position $n > m$. We
 10209 might hope for some dynamic programming solution, as in sequence labeling (§ 7.3). But
 10210 the Viterbi algorithm and its relatives rely on a Markov decomposition of the objective
 10211 function into a sum of local scores: for example, scores can consider locally adjacent tags
 10212 (y_m, y_{m-1}), but not the entire tagging history $y_{1:m}$. This decomposition is not applicable
 10213 to recurrent neural networks, because the hidden state $h_m^{(t)}$ is impacted by the entire his-
 10214 tory $w_{1:m}^{(t)}$; this sensitivity to long-range context is precisely what makes recurrent neural
 10215 networks so effective.¹⁰ In fact, it can be shown that decoding from any recurrent neural
 10216 network is NP-complete (Siegelmann and Sontag, 1995; Chen et al., 2018).

10217 **Beam search** Beam search is a general technique for avoiding search errors when ex-
 10218 haustive search is impossible; it was first discussed in § 11.3.1.4. Beam search can be
 10219 seen as a variant of the incremental decoding algorithm sketched in Equation 18.50, but
 10220 at each step m , a set of K different hypotheses are kept on the beam. For each hypothesis
 10221 $k \in \{1, 2, \dots, K\}$, 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
 10222 the current hidden state $h_k^{(t)}$. At each step in the beam search, the K top-scoring children
 10223 of each hypothesis currently on the beam are “expanded”, and the beam is updated. For
 10224 a detailed description of beam search for RNN decoding, see Graves (2012).

10225 **Learning and search** Conventionally, the learning algorithm is trained to predict the
 10226 right token in the translation, conditioned on the translation history being correct. But
 10227 if decoding must be approximate, then we might do better by modifying the learning
 10228 algorithm to be robust to errors in the translation history. **Scheduled sampling** does this
 10229 by training on histories that sometimes come from the ground truth, and sometimes come
 10230 from the model’s own output (Bengio et al., 2015).¹¹ As training proceeds, the training
 10231 wheels come off: we increase the fraction of tokens that come from the model rather than
 10232 the ground truth. Another approach is to train on an objective that relates directly to beam
 10233 search performance (Wiseman et al., 2016). **Reinforcement learning** has also been applied
 10234 to decoding of RNN-based translation models, making it possible to directly optimize
 10235 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.

10236 18.5 Training towards the evaluation metric

10237 In likelihood-based training, the objective is to maximize the probability of a parallel
 10238 corpus. However, translations are not evaluated in terms of likelihood: metrics like BLEU
 10239 consider only the correctness of a single output translation, and not the range of prob-
 10240 abilities that the model assigns. It might therefore be better to train translation models
 10241 to achieve the highest BLEU score possible — to the extent that we believe BLEU mea-
 10242 sures translation quality. Unfortunately, BLEU and related metrics are not friendly for
 10243 optimization: they are discontinuous, non-differentiable functions of the parameters of
 10244 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,

$$\hat{\mathbf{w}}^{(t)} = \operatorname{argmax}_{\mathbf{w}^{(t)}} \Psi(\mathbf{w}^{(t)}, \mathbf{w}^{(s)}; \theta) \quad [18.51]$$

$$\hat{\theta} = \operatorname{argmin}_{\theta} \Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(s)}) \quad [18.52]$$

10245 However, identifying the top-scoring translation $\hat{\mathbf{w}}^{(t)}$ is usually intractable, as described
 10246 in the previous section. In **minimum error-rate training (MERT)**, $\hat{\mathbf{w}}^{(t)}$ is selected from a
 10247 set of candidate translations $\mathcal{Y}(\mathbf{w}^{(s)})$; this is typically a strict subset of all possible transla-
 10248 tions, so that it is only possible to optimize an approximation to the true error rate (Och
 10249 and Ney, 2003).

A further issue is that the objective function in Equation 18.52 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,

$$R(\theta) = E_{\hat{\mathbf{w}}^{(t)} | \mathbf{w}^{(s)}; \theta} [\Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)})] \quad [18.53]$$

$$= \sum_{\hat{\mathbf{w}}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})} p(\hat{\mathbf{w}}^{(t)} | \mathbf{w}^{(s)}) \times \Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)}). \quad [18.54]$$

10250 **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,

$$\tilde{p}(\mathbf{w}^{(t)}; \theta, \alpha) \propto \left(p(\mathbf{w}^{(t)} | \mathbf{w}^{(s)}; \theta) \right)^\alpha \quad [18.55]$$

$$\tilde{R}(\theta) = \sum_{\hat{\mathbf{w}}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})} \tilde{p}(\hat{\mathbf{w}}^{(t)} | \mathbf{w}^{(s)}; \alpha, \theta) \times \Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)}) \quad [18.56]$$

10251 where $\mathcal{Y}(\mathbf{w}^{(s)})$ is now the set of *all* possible translations for $\mathbf{w}^{(s)}$. Exponentiating the prob-
 10252 abilities in this way is known as **annealing** (Smith and Eisner, 2006). When $\alpha = 1$, then
 10253 $\tilde{R}(\boldsymbol{\theta}) = R(\boldsymbol{\theta})$; when $\alpha = \infty$, then $\tilde{R}(\boldsymbol{\theta})$ is equivalent to the sum of the errors of the maxi-
 10254 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),

$$\tilde{R}(\boldsymbol{\theta}) \approx \frac{1}{Z} \sum_{k=1}^K \tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)}; \boldsymbol{\theta}, \alpha) \times \Delta(\mathbf{w}_k^{(t)}, \mathbf{w}^{(t)}) \quad [18.57]$$

$$Z = \sum_{k=1}^K \tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)}; \boldsymbol{\theta}, \alpha). \quad [18.58]$$

10255 Shen et al. (2016) report that performance plateaus at $K = 100$ for minimum risk training
 10256 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** $q(\mathbf{w}^{(s)})$. 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**, $\omega_k = \frac{\tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)})}{q(\mathbf{w}_k^{(t)}; \mathbf{w}^{(s)})}$. 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,

$$\mathbf{w}_k^{(t)} \sim q(\mathbf{w}^{(s)}) \quad [18.59]$$

$$\omega_k = \frac{\tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)})}{q(\mathbf{w}_k^{(t)}; \mathbf{w}^{(s)})} \quad [18.60]$$

$$\tilde{R}(\boldsymbol{\theta}) \approx \frac{1}{\sum_{k=1}^K \omega_k} \sum_{k=1}^K \omega_k \times \Delta(\mathbf{w}_k^{(t)}, \mathbf{w}^{(t)}). \quad [18.61]$$

10257 Importance sampling will generally give a more accurate approximation than uniform
 10258 sampling. The only formal requirement is that the proposal assigns non-zero probability
 10259 to every $\mathbf{w}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})$. For more on importance sampling and related methods, see
 10260 Robert and Casella (2013).

10261 **Additional resources**

10262 A complete textbook on machine translation is available from Koehn (2009). While this
10263 book precedes recent work on neural translation, a more recent draft chapter on neural
10264 translation models is also available (Koehn, 2017). Neubig (2017) provides a comprehen-
10265 sive tutorial on neural machine translation, starting from first principles. The course notes
10266 from Cho (2015) are also useful. Several neural machine translation systems are avail-
10267 able: `lamt ram` is an implementation of neural machine translation in the `dynet` (Neubig
10268 et al., 2017); `OpenNMT` (Klein et al., 2017) and `FairSeq` are an implementation primar-
10269 ily in `Torch`; `tensor2tensor` is an implementation of several of the Google translation
10270 models in `tensorflow` (Abadi et al., 2016).

10271 Literary translation is especially challenging, even for expert human translators. Mes-
10272 sud (2014) describes some of these issues in her review of an English translation of *L'étranger*,
10273 the 1942 French novel by Albert Camus.¹² She compares the new translation by Sandra
10274 Smith against earlier translations by Stuart Gilbert and Matthew Ward, focusing on the
10275 difficulties presented by a single word in the first sentence:

10276 Then, too, Smith has reconsidered the book's famous opening. Camus's
10277 original is deceptively simple: "*Aujourd'hui, maman est morte.*" Gilbert influ-
10278 enced generations by offering us "Mother died today"—inscribing in Meur-
10279 sault [the narrator] from the outset a formality that could be construed as
10280 heartlessness. But *maman*, after all, is intimate and affectionate, a child's name
10281 for his mother. Matthew Ward concluded that it was essentially untranslatable
10282 ("mom" or "mummy" being not quite apt), and left it in the original French:
10283 "Maman died today." There is a clear logic in this choice; but as Smith has
10284 explained, in an interview in *The Guardian*, *maman* "didn't really tell the reader
10285 anything about the connotation." She, instead, has translated the sentence as
10286 "My mother died today."

10287 I chose "My mother" because I thought about how someone would
10288 tell another person that his mother had died. Meursault is speaking
10289 to the reader directly. "My mother died today" seemed to me the
10290 way it would work, and also implied the closeness of "maman" you
10291 get in the French.

10292 Elsewhere in the book, she has translated *maman* as "mama"—again, striving
10293 to come as close as possible to an actual, colloquial word that will carry the
10294 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/>.

10295 The passage is a useful reminder that while the quality of machine translation has
 10296 improved dramatically in recent years, expert human translations draw on considerations
 10297 that are beyond the ken of any known computational approach.

10298 **Exercises**

10299 1. Using Google translate or another online service, translate the following example
 10300 into two different languages of your choice:

10301 (18.4) It is not down on any map; true places never are.

10302 Then translate each result back into English. Which is closer to the original? Can
 10303 you explain the differences?

10304 2. Compute the unsmoothed n -gram precisions $p_1 \dots p_4$ for the two back-translations
 10305 in the previous problem. Your n -gramsshould include punctuation, and should seg-
 10306 ment conjunctions like *it's* into two tokens.

10307 3. You are given the following dataset of translations from “simple” to “difficult” En-
 10308 glish:

10309 (18.5) a. *Kids like cats.*
 Children adore felines.

10310 b. *Cats hats.*
 Felines fedoras.

10311 Estimate a word-to-word statistical translation model from simple English (source)
 10312 to difficult English (target), using the expectation-maximization as described in § 18.2.2.
 10313 Compute two iterations of the algorithm by hand, starting from a uniform transla-
 10314 tion model, and using the simple alignment model $p(a_m \mid m, M^{(s)}, M^{(t)}) = \frac{1}{M^{(t)}}$.
 10315 Hint: in the final M-step, you will want to switch from fractions to decimals.

10316 4. Building on the previous problem, what will be the converged translation proba-
 10317 bility table? Can you state a general condition about the data, under which this
 10318 translation model will fail in the way that it fails here?

10319 5. Propose a simple alignment model that would make it possible to recover the correct
 10320 translation probabilities from the toy dataset in the previous two problems.

10321 6. Give a synchronized derivation (§ 18.2.4) for the Spanish-English translation,

- 10322 (18.6) *El pez enojado atacado.*
 The fish angry attacked.
 10323 The angry fish attacked.

10324 As above, the second line shows a word-for-word gloss, and the third line shows
 10325 the desired translation. Use the synchronized production rule in [18.22], and design
 10326 the other production rules necessary to derive this sentence pair. You may derive
 10327 (*atacado*, *attacked*) directly from VP.

- 10328 7. Let $\ell_{m+1}^{(t)}$ represent the loss at word $m+1$ of the target, and let $\mathbf{h}_n^{(s)}$ represent the hid-
 10329 den state at word n of the source. Write the expression for the derivative $\frac{\partial \ell_{m+1}^{(t)}}{\partial \mathbf{h}_n^{(s)}}$ in the
 10330 sequence-to-sequence translation model expressed in Equations [18.29-18.32]. You
 10331 may assume that both the encoder and decoder are one-layer LSTMs. In general,
 10332 how many terms are on the shortest path from $\ell_{m+1}^{(t)}$ to $\mathbf{h}_n^{(s)}$?
- 10333 8. Now consider the neural attentional model from § 18.3.1, with sigmoid attention.
 10334 The derivative $\frac{\partial \ell_{m+1}^{(t)}}{\partial z_n}$ is the sum of many paths through the computation graph;
 10335 identify the shortest such path. You may assume that the initial state of the decoder
 10336 recurrence $\mathbf{h}_0^{(t)}$ is *not* tied to the final state of the encoder recurrence $\mathbf{h}_{M^{(s)}}$.
- 10337 9. Apply byte-pair encoding for the vocabulary *it*, *unit*, *unite*, until no bigram appears
 10338 more than once.
- 10339 10. Hand-design an attentional recurrent translation model that simply copies the input
 10340 from the source to the target. You may assume an arbitrarily large hidden state, and
 10341 you may assume that there is a finite maximum input length M . Specify all the
 10342 weights such that the maximum probability translation of any source is the source
 10343 itself. Hint: it is simplest to use a simple Elman-recurrence $\mathbf{h}_m = f(\Theta \mathbf{h}_{m-1} + \mathbf{x}_m)$
 10344 rather than an LSTM.
- 10345 11. This problem relates to the complexity of machine translation. Suppose you have
 10346 an oracle that returns the list of words to include in the translation, so that your
 10347 only task is to order the words. Furthermore, suppose that the scoring function
 10348 over orderings is a sum over bigrams, $\sum_{m=1}^M \psi(\mathbf{w}_m^{(t)}, \mathbf{w}_{m-1}^{(t)})$. Show that the problem
 10349 of finding the optimal translation is NP-complete, by reduction from a well-known
 10350 problem.

10351

Chapter 19

10352

Text generation

10353 In many of the most interesting problems in natural language processing, language is
10354 the output. The previous chapter described the specific case of machine translation, but
10355 there are many other applications, from summarization of research articles, to automated
10356 journalism, to dialogue systems. This chapter emphasizes three main scenarios: data-to-
10357 text, in which text is generated to explain or describe a structured record or unstructured
10358 perceptual input; text-to-text, which typically involves fusing information from multiple
10359 linguistic sources into a single coherent summary; and dialogue, in which text is generated
10360 as part of an interactive conversation with one or more human participants.

10361

19.1 Data-to-text generation

10362 In data-to-text generation, the input ranges from structured records, such as the descrip-
10363 tion of an weather forecast (as shown in Figure 19.1), to unstructured perceptual data,
10364 such as a raw image or video; the output may be a single sentence, such as an image cap-
10365 tion, or a multi-paragraph argument. Despite this diversity of conditions, all data-to-text
10366 systems share some of the same challenges (Reiter and Dale, 2000):

- 10367 • determining what parts of the data to describe;
10368 • planning a presentation of this information;
10369 • **lexicalizing** the data into words and phrases;
10370 • organizing words and phrases into well-formed sentences and paragraphs.

10371 The earlier stages of this process are sometimes called **content selection** and **text plan-**
10372 **ning**; the later stages are often called **surface realization**.

10373 Early systems for data-to-text generation were modular, with separate software com-
10374 ponents for each task. Artificial intelligence **planning** algorithms can be applied to both

Temperature				Cloud sky cover	
<i>time</i>	<i>min</i>	<i>mean</i>	<i>max</i>	<i>time</i>	<i>percent (%)</i>
06:00-21:00	9	15	21	06:00-09:00	25-50
Wind speed				Wind direction	
<i>time</i>	<i>min</i>	<i>mean</i>	<i>max</i>	<i>time</i>	<i>mode</i>
06:00-21:00	15	20	30	06:00-21:00	S

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 (adapted from Konstas and Lapata, 2013).

10375 the high-level information structure and the organization of individual sentences, ensur-
 10376 ing that communicative goals are met (McKeown, 1992; Moore and Paris, 1993). Surface
 10377 realization can be performed by grammars or templates, which link specific types of data
 10378 to candidate words and phrases. A simple example template is offered by Wiseman et al.
 10379 (2017), for generating descriptions of basketball games:

10380 (19.1) The <team1>(<wins1>-<losses1>) defeated the <team2>(<wins2>-<losses2>),
 10381 <pts1>-<pts2>.
 10382 The New York Knicks (45-5) defeated the Boston Celtics (11-38), 115-79.

10383 For more complex cases, it may be necessary to apply morphological inflections such as
 10384 pluralization and tense marking — even in the simple example above, languages such
 10385 as Russian would require case marking suffixes for the team names. Such inflections can
 10386 be applied as a postprocessing step. Another difficult challenge for surface realization is
 10387 the generation of varied **referring expressions** (e.g., *The Knicks, New York, they*), which is
 10388 critical to avoid repetition. As discussed in § 16.2.1, the form of referring expressions is
 10389 constrained by the discourse and information structure.

10390 An example at the intersection of rule-based and statistical techniques is the Nitrogen
 10391 system (Langkilde and Knight, 1998). The input to Nitrogen is an abstract meaning rep-
 10392 resentation (AMR; see § 13.3) of semantic content to be expressed in a single sentence. In
 10393 data-to-text scenarios, the abstract meaning representation is the output of a higher-level
 10394 text planning stage. A set of rules then converts the abstract meaning representation into
 10395 various sentence plans, which may differ in both the high-level structure (e.g., active ver-
 10396 sus passive voice) as well as the low-level details (e.g., word and phrase choice). Some
 10397 examples are shown in Figure 19.2. To control the combinatorial explosion in the number
 10398 of possible realizations for any given meaning, the sentence plans are unified into a single
 10399 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} . Specifically, let 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

$$(\textit{cloudy}, \textit{cloud-sky-cover:percent}). \quad [19.2]$$

In general, given an observed set of alignments, the score for a generation can be

10422 written as sum of local scores (Angeli et al., 2010):

$$\Psi(\mathbf{w}, \mathbf{y}; \theta) = \sum_{m=1}^M \psi_{w,y}(\mathbf{w}_m, \mathbf{y}_{z_m}) + \psi_w(w_m, w_{m-1}) + \psi_z(z_m, z_{m-1}), \quad [19.3]$$

10423 where ψ_w can represent a bigram language model, and ψ_z can be tuned to reward coherence,
 10424 such as the use of related records in nearby words.¹ The parameters of this model
 10425 could be learned from labeled data $\{(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})\}_{i=1}^N$. However, while several datasets
 10426 include structured records and natural language text (Barzilay and McKeown, 2005; Chen
 10427 and Mooney, 2008; Liang and Klein, 2009), the alignments between text and records are
 10428 usually not available.² One solution is to model the problem probabilistically, treating the
 10429 alignment as a latent variable (Liang et al., 2009; Konstas and Lapata, 2013). The model
 10430 can then be estimated using expectation maximization or sampling (see chapter 5).

10431 19.1.2 Neural data-to-text generation

10432 The **encoder-decoder model** and **neural attention** were introduced in § 18.3 as methods
 10433 for neural machine translation. They can also be applied to data-to-text generation, with
 10434 the data acting as the source language (Mei et al., 2016). In neural machine translation,
 10435 the attention mechanism linked words in the source to words in the target; in data-to-
 10436 text generation, the attention mechanism can link each part of the generated text back
 10437 to a record in the data. The biggest departure from translation is in the encoder, which
 10438 depends on the form of the data.

10439 19.1.2.1 Data encoders

10440 In some types of structured records, all values are drawn from discrete sets. For example,
 10441 the birthplace of an individual is drawn from a discrete set of possible locations; the diag-
 10442 nosis and treatment of a patient are drawn from an exhaustive list of clinical codes (John-
 10443 son et al., 2016). In such cases, vector embeddings can be estimated for each field and
 10444 possible value: for example, a vector embedding for the field BIRTHPLACE, and another
 10445 embedding for the value BERKELEY_CALIFORNIA (Bordes et al., 2011). The table of such
 10446 embeddings serves as the encoding of a structured record (He et al., 2017). It is also possi-
 10447 ble to compress the entire table into a single vector representation, by **pooling** across the
 10448 embeddings of each field and value (Lebret et al., 2016).

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).

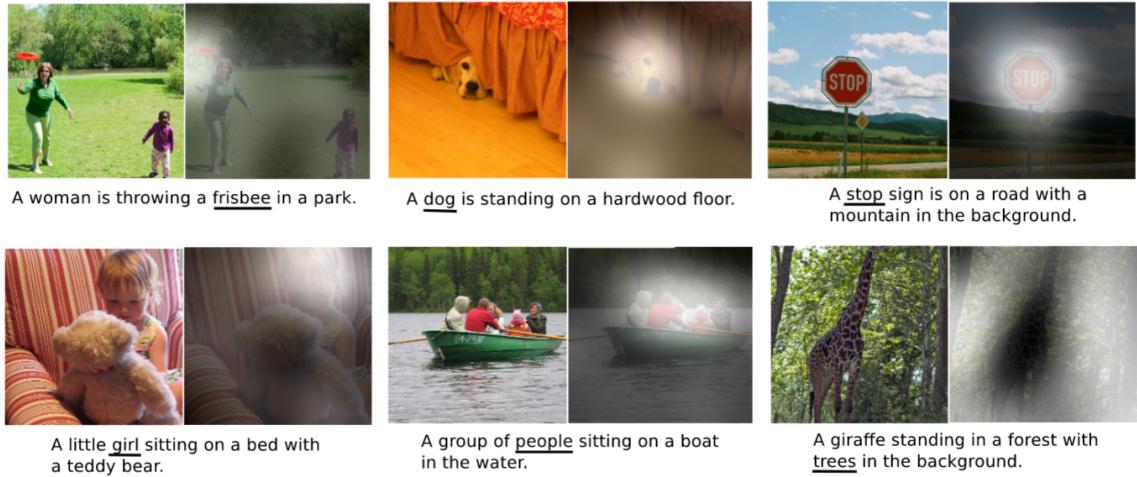


Figure 19.3: Examples of the image captioning task, with attention masks shown for each of the underlined words. From Xu et al. (2015). [todo: permission]

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 et al., 2016):

```
PASS(arg1 = PURPLE6, arg2 = PURPLE3)
KICK(arg1 = PURPLE3)
BADPASS(arg1 = PURPLE3, arg2 = PINK9).
```

10449 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),
 10450 e.g.,
 10451

$$\mathbf{X} = [\mathbf{u}_{\text{PASS}}, \mathbf{u}_{\text{arg1}}, \mathbf{u}_{\text{PURPLE6}}, \mathbf{u}_{\text{arg2}}, \mathbf{u}_{\text{PURPLE3}}]. \quad [19.4]$$

10452 This encoding can then act as the input layer for a recurrent neural network, yielding a
 10453 sequence of vector representations $\{\mathbf{z}_r\}_{r=1}^R$, where r indexes over records. Interestingly,
 10454 this sequence-based approach can work even in cases where there is no natural ordering
 10455 over the records, such as the weather data in Figure 19.1 (Mei et al., 2016).

10456 **Images** Another flavor of data-to-text generation is the generation of text captions for
 10457 images. Examples from this task are shown in Figure 19.3. Images are naturally represented as tensors: a color image of 320×240 pixels would be stored as a tensor with
 10458 $320 \times 240 \times 3$ intensity values. The dominant approach to image classification is to en-
 10459 code images as vectors using a combination of convolution and pooling (Krizhevsky et al.,
 10460

10461 2012). Chapter 3 explains how to use convolutional networks for text; for images, convolution
 10462 is applied across the vertical, horizontal, and color dimensions. By pooling the re-
 10463 sults of successive convolutions, the image is converted to a vector representation, which
 10464 can then be fed directly into the decoder as the initial state (Vinyals et al., 2015), just as
 10465 in the sequence-to-sequence translation model (see § 18.3). Alternatively, one can apply
 10466 a set of convolutional networks, yielding vector representations for different parts of the
 10467 image, which can then be combined using neural attention (Xu et al., 2015).

10468 **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 , an attention vector over the data can be computed using the same techniques as in machine translation (see § 18.3.1). When generating word m of the output, attention is computed over the records,

$$\psi_\alpha(m, r) = \beta_\alpha \cdot f(\Theta_\alpha[\mathbf{h}_m; \mathbf{z}_r]) \quad [19.5]$$

$$\boldsymbol{\alpha}_m = g([\psi_\alpha(m, 1), \psi_\alpha(m, 2), \dots, \psi_\alpha(m, R)]) \quad [19.6]$$

$$\mathbf{c}_m = \sum_{r=1}^R \alpha_{m \rightarrow r} \mathbf{z}_r, \quad [19.7]$$

10469 where f is an elementwise nonlinearity such as tanh or ReLU, and g is either softmax or
 10470 elementwise sigmoid. The weighted sum \mathbf{c}_m can then be included in the recurrent update
 10471 to the decoder state, or in the emission probabilities, as described in § 18.3.1. Figure 19.4
 10472 shows the attention to components of a weather record, while generating the text shown
 10473 on the x -axis.

10474 Adapting this architecture to image captioning is straightforward. A convolutional
 10475 neural networks is applied to a set of image locations, and the output at each location ℓ is
 10476 represented with a vector \mathbf{z}_ℓ . Attention can then be computed over the image locations,
 10477 as shown in the right panels of each pair of images in Figure 19.3.

10478 Various modifications to this basic mechanism have been proposed. In **coarse-to-fine**
 10479 **attention** (Mei et al., 2016), each record receives a global attention $a_r \in [0, 1]$, which is in-
 10480 dependent of the decoder state. This global attention, which represents the overall impor-
 10481 tance of the record, is multiplied with the decoder-based attention scores, before comput-
 10482 ing the final normalized attentions. In **structured attention**, the attention vector $\boldsymbol{\alpha}_{m \rightarrow \cdot}$ can
 10483 include structural biases, which can favor assigning higher attention values to contiguous
 10484 segments or to dependency subtrees (Kim et al., 2017). Structured attention vectors can
 10485 be computed by running the forward-backward algorithm to obtain marginal attention
 10486 probabilities (see § 7.5.3.3). Because each step in the forward-backward algorithm is dif-
 10487 ferentiable, it can be encoded in a computation graph, and end-to-end learning can be
 10488 performed by backpropagation.

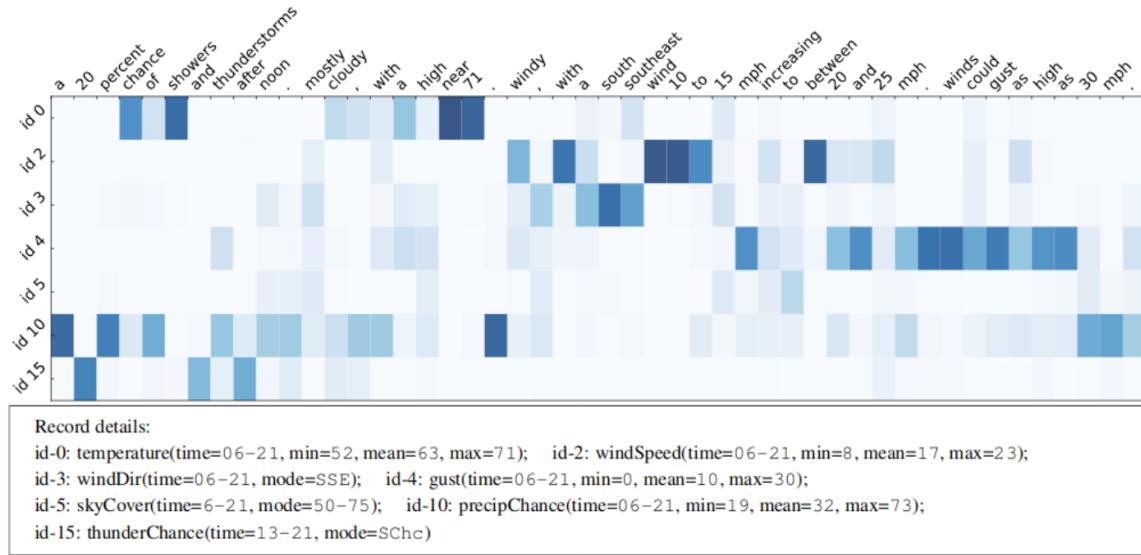


Figure 19.4: Neural attention in text generation. Figure from Mei et al. (2016).[todo: permission]

10489 19.1.2.3 Decoder

10490 Given the encoding, the decoder can function just as in neural machine translation (see
 10491 § 18.3.1), using the attention-weighted encoder representation in the decoder recurrence
 10492 and/or output computation. As in machine translation, beam search can help to avoid
 10493 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,

$$p(w^{(t)} | w_{1:m-1}^{(t)}, \mathbf{Z}, s_m) = \begin{cases} \text{SoftMax}(\beta_{w^{(t)}} \cdot h_{m-1}^{(t)}), & s_m = \text{gen} \\ \sum_{r=1}^R \delta(w_r^{(s)} = w^{(t)}) \times \alpha_{m \rightarrow r}, & s_m = \text{copy}, \end{cases} \quad [19.8]$$

10494 where $\delta(w_r^{(s)} = w^{(t)})$ is an indicator function, taking the value 1 iff the text of the record
 10495 $w_r^{(s)}$ is identical to the target word $w^{(t)}$. The probability of copying record r from the source

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.

10496 is $\delta(s_m = \text{copy}) \times \alpha_{m \rightarrow r}$, the product of the copy probability by the local attention. Note
 10497 that in this model, the attention weights α_m are computed from the *previous* decoder state
 10498 \mathbf{h}_{m-1} . The computation graph therefore remains a feedforward network, with recurrent
 10499 paths such as $\mathbf{h}_{m-1}^{(t)} \rightarrow \alpha_m \rightarrow w_m^{(t)} \rightarrow \mathbf{h}_m^{(t)}$.

10500 To facilitate end-to-end training, the switching variable s_m can be represented by a
 10501 gate π_m , which is computed from a two-layer feedforward network, whose input consists
 10502 of the concatenation of the decoder state $\mathbf{h}_{m-1}^{(t)}$ and the attention-weighted representation
 10503 of the data, $c_m = \sum_{r=1}^R \alpha_{m \rightarrow r} z_r$,

$$\pi_m = \sigma(\Theta^{(2)} f(\Theta^{(1)}[\mathbf{h}_{m-1}^{(t)}; c_m])). \quad [19.9]$$

The full generative probability at token m is then,

$$p(w^{(t)} | \mathbf{w}_{1:m}^{(t)}, \mathbf{Z}) = \pi_m \times \underbrace{\frac{\exp \beta_{w^{(t)}} \cdot \mathbf{h}_{m-1}^{(t)}}{\sum_{j=1}^V \exp \beta_j \cdot \mathbf{h}_{m-1}^{(t)}}}_{\text{generate}} + (1 - \pi_m) \times \underbrace{\sum_{r=1}^R \delta(w_r^{(s)} = w^{(t)}) \times \alpha_{m \rightarrow r}}_{\text{copy}}. \quad [19.10]$$

10504 19.2 Text-to-text generation

10505 Text-to-text generation includes problems of summarization and simplification:

- 10506 • reading a novel and outputting a paragraph-long summary of the plot;⁴
- 10507 • reading a set of blog posts about politics, and outputting a bullet list of the various
 10508 issues and perspectives;
- 10509 • reading a technical research article about the long-term health consequences of drink-
 10510 ing kombucha, and outputting a summary of the article in language that non-experts
 10511 can understand.

10512 These problems can be approached in two ways: through the encoder-decoder architec-
 10513 ture discussed in the previous section, or by operating directly on the input text.

10514 19.2.1 Neural abstractive summarization

10515 **Sentence summarization** is the task of shortening a sentence while preserving its mean-
 10516 ing, as in the following examples (Knight and Marcu, 2000; Rush et al., 2015):

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.

- 10517 (19.2) The documentation is typical of Epson quality: excellent.
 10518 Documentation is excellent.
 10519
- 10520 (19.3) Russian defense minister Ivanov called sunday for the creation of a joint front for
 10521 combating global terrorism.
 10522 Russia calls for joint front against terrorism.
 10523
- 10524 Sentence summarization is closely related to **sentence compression**, in which the sum-
 10525 mary is produced by deleting words or phrases from the original (Clarke and Lapata,
 10526 2008). But as shown in (19.3), a sentence summary can also introduce new words, such as
 10527 *against*, which replaces the phrase *for combatting*.
 10528 Sentence summarization can be treated as a machine translation problem, using the at-
 10529 tentional encoder-decoder translation model discussed in § 18.3.1 (Rush et al., 2015). The
 10530 longer sentence is encoded into a sequence of vectors, one for each token. The decoder
 10531 then computes attention over these vectors when updating its own recurrent state. As
 10532 with data-to-text generation, it can be useful to augment the encoder-decoder model with
 10533 the ability to copy words directly from the source. Rush et al. (2015) train this model by
 10534 building four million sentence pairs from news articles. In each pair, the longer sentence is
 10535 the first sentence of the article, and the summary is the article headline. Sentence summa-
 10536 rization can also be trained in a semi-supervised fashion, using a probabilistic formulation
 10537 of the encoder-decoder model called a **variational autoencoder** (Miao and Blunsom, 2016,
 10538 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,

$$\ell_m = \sum_{n=1}^{M^{(s)}} \min(\alpha_{m \rightarrow n}, t_{m \rightarrow n}). \quad [19.12]$$

- 10539 This loss will be low if α_m assigns little attention to words that already have large values in
 10540 t_m . Coverage loss is similar to the concept of **marginal relevance**, in which the reward for
 10541 adding new content is proportional to the extent to which it increases the overall amount
 10542 of information conveyed by the summary (Carbonell and Goldstein, 1998).

10543 **19.2.2 Sentence fusion for multi-document summarization**

10544 In **multi-document summarization**, the goal is to produce a summary that covers the
 10545 content of several documents (McKeown et al., 2002). One approach to this challenging
 10546 problem is to identify sentences across multiple documents that relate to a single theme,
 10547 and then to fuse them into a single sentence (Barzilay and McKeown, 2005). As an exam-
 10548 ple, consider the following two sentences (McKeown et al., 2010):

- 10549 (19.4) Palin actually turned against the bridge project only after it became a national
 10550 symbol of wasteful spending.
- 10551 (19.5) Ms. Palin supported the bridge project while running for governor, and aban-
 10552 doned it after it became a national scandal.

10553 An *intersection* preserves only the content that is present in both sentences:

- 10554 (19.6) Palin turned against the bridge project after it became a national scandal.

10555 A *union* includes information from both sentences:

- 10556 (19.7) Ms. Palin supported the bridge project while running for governor, but turned
 10557 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),

$$\max_{\mathbf{y}} \sum_{i,j,r} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) \times y_{i,j,r} \quad [19.13]$$

$$\text{s.t. } \mathbf{y} \in \mathcal{C}, \quad [19.14]$$

10558 where the variable $y_{i,j,r} \in \{0, 1\}$ indicates whether there is an edge from i to j of type r ,
 10559 the score of this edge is $\psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})$, and \mathcal{C} is a set of constraints, which ensures that \mathbf{y}
 10560 forms a valid dependency graph. As usual, \mathbf{w} is the list of words in the graph, and $\boldsymbol{\theta}$ is a
 10561 vector of parameters. The score $\psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})$ reflects the “importance” of the modifier
 10562 j to the overall meaning: in intersective fusion, this score indicates the extent to which
 10563 the content in this edge is expressed in all sentences; in union fusion, the score indicates
 10564 whether the content in the edge is expressed in any sentence. The constraint set \mathcal{C} can
 10565 impose additional linguistic constraints: for example, ensuring that coordinated nouns
 10566 are sufficiently similar. The resulting tree must then be **linearized** into a sentence. Lin-
 10567 earization is like the inverse of dependency parsing: instead of parsing from a sequence

of tokens into a tree, we must convert the tree back into a sequence of tokens. This is typically done by generating a set of candidate linearizations, and choosing the one with the highest score under a language model (Langkilde and Knight, 1998; Song et al., 2016).

10571 19.3 Dialogue

Dialogue systems are capable of conversing with a human interlocutor, often to perform some task (Grosz, 1979), but sometimes just to chat (Weizenbaum, 1966). While research on dialogue systems goes back several decades (Carbonell, 1970; Winograd, 1972), commercial systems such as Alexa and Siri have recently brought this technology into widespread use. Nonetheless, there is a significant gap between research and practice: many practical dialogue systems remain scripted and inflexible, while research systems emphasize abstractive text generation, “on-the-fly” decision making, and probabilistic reasoning about the user’s intentions.

10580 19.3.1 Finite-state and agenda-based dialogue systems

Finite-state automata were introduced in chapter 9 as a formal model of computation, in which string inputs and outputs are linked to transitions between a finite number of discrete states. This model naturally fits simple task-oriented dialogues, such as the one shown in the left panel of Figure 19.5. This (somewhat frustrating) dialogue can be represented with a finite-state transducer, as shown in the right panel of the figure. The accepting state is reached only when the two needed pieces of information are provided, and the human user confirms that the order is correct. In this simple scenario, the TOPPING and ADDRESS are the two **slots** associated with the activity of ordering a pizza, which is called a **frame**. Frame representations can be hierarchical: for example, an ADDRESS could have slots of its own, such as STREET and CITY.

In the example dialogue in Figure 19.5, the user provides the precise inputs that are needed in each turn (e.g., *anchovies*; *the College of Computing building*). Some users may prefer to communicate more naturally, with phrases like *I'd, uh, like some anchovies please*. One approach to handling such utterances is to design a custom grammar, with non-terminals for slots such as TOPPING and LOCATION. However, context-free parsing of unconstrained speech input is challenging. A more lightweight alternative is BIO-style sequence labeling (see § 8.3), e.g.:

- 10598 (19.9) *I'd like anchovies , and please bring it to the College of Computing*
O O B-TOPPING O O O O O O B-ADDR I-ADDR I-ADDR I-ADDR I-ADDR
10599 *Building .*
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?
 ...

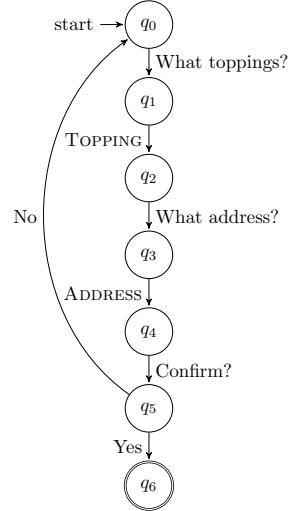


Figure 19.5: An example dialogue and the associated finite-state model. In the finite-state model, SMALL CAPS indicates that the user must provide information of this type in their answer.

10600 The tagger can be driven by a bi-directional recurrent neural network, similar to recurrent
 10601 approaches to semantic role labeling described in § 13.2.3.

10602 The input in (19.9) could not be handled by the finite-state system from Figure 19.5,
 10603 which forces the user to provide the topping first, and then the location. In this sense,
 10604 the **initiative** is driven completely by the system. **Agenda-based dialogue systems** ex-
 10605 tend finite-state architectures by attempting to recognize all slots that are filled by the
 10606 user’s reply, thereby handling these more complex examples. Agenda-based systems dy-
 10607 namically pose additional questions until the frame is complete (Bobrow et al., 1977; Allen
 10608 et al., 1995; Rudnicky and Xu, 1999). Such systems are said to be **mixed-initiative**, because
 10609 both the user and the system can drive the direction of the dialogue.

10610 19.3.2 Markov decision processes

10611 The task of dynamically selecting the next move in a conversation is known as **dialogue**
 10612 **management**. This problem can be framed as a **Markov decision process**, which is a
 10613 theoretical model that includes a discrete set of states, a discrete set of actions, a function
 10614 that computes the probability of transitions between states, and a function that computes
 10615 the cost or reward of action-state pairs. Let’s see how each of these elements pertains to
 10616 the pizza ordering dialogue system.

- 10617 • Each state is a tuple of information about whether the topping and address are

10618 known, and whether the order has been confirmed. For example,

(KNOWN TOPPING, UNKNOWN ADDRESS, NOT CONFIRMED) [19.15]

10619 is a possible state. Any state in which the pizza order is confirmed is a terminal
10620 state, and the Markov decision process stops after entering such a state.

- 10621 • The set of actions includes querying for the topping, querying for the address, and
10622 requesting confirmation. Each action induces a probability distribution over states,
10623 $p(s_t | a_t, s_{t-1})$. For example, requesting confirmation of the order is not likely to
10624 result in a transition to the terminal state if the topping is not yet known. This
10625 probability distribution over state transitions may be learned from data, or it may
10626 be specified in advance.
- 10627 • Each state-action-state tuple earns a reward, $r_a(s_t, s_{t+1})$. In the context of the pizza
10628 ordering system, a simple reward function would be,

$$r_a(s_t, s_{t-1}) = \begin{cases} 0, & a = \text{CONFIRM}, s_t = (*, *, \text{CONFIRMED}) \\ -10, & a = \text{CONFIRM}, s_t = (*, *, \text{NOT CONFIRMED}) \\ -1, & a \neq \text{CONFIRM} \end{cases} \quad [19.16]$$

10629 This function assigns zero reward for successful transitions to the terminal state, a
10630 large negative reward to a rejected request for confirmation, and a small negative re-
10631 ward for every other type of action. The system is therefore rewarded for reaching
10632 the terminal state in few steps, and penalized for prematurely requesting confirma-
10633 tion.

10634 In a Markov decision process, a **policy** is a function $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maps from states to
10635 actions (see § 15.2.4.3). The value of a policy is the expected sum of discounted rewards,
10636 $E_\pi[\sum_{t=1}^T \gamma^t r_{a_t}(s_t, s_{t+1})]$, where γ is the discount factor, $\gamma \in [0, 1)$. Discounting has the
10637 effect of emphasizing rewards that can be obtained immediately over less certain rewards
10638 in the distant future.

10639 An optimal policy can be obtained by dynamic programming, by iteratively updating
10640 the **value function** $V(s)$, which is the expectation of the cumulative reward from s under
10641 the optimal action a ,

$$V(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s' | s, a)[r_a(s, s') + \gamma V(s')]. \quad [19.17]$$

10642 The value function $V(s)$ is computed in terms of $V(s')$ for all states $s' \in \mathcal{S}$. A series
10643 of iterative updates to the value function will eventually converge to a stationary point.
10644 This algorithm is known as **value iteration**. Given the converged value function $V(s)$, the

10645 optimal action at each state is the argmax,

$$\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s' | s, a)[r_a(s, s') + \gamma V(s')]. \quad [19.18]$$

10646 Value iteration and related algorithms are described in detail by Sutton and Barto (1998).
 10647 For applications to dialogue systems, see Levin et al. (1998) and Walker (2000).

10648 The Markov decision process framework assumes that the current state of the dialogue
 10649 is known. In reality, the system may misinterpret the user’s statements — for example,
 10650 believing that a specification of the delivery location (PEACHTREE) is in fact a specification
 10651 of the topping (PEACHES). In a **partially observable Markov decision process (POMDP)**,
 10652 the system receives an *observation* o , which is probabilistically conditioned on the state,
 10653 $p(o | s)$. It must therefore maintain a distribution of beliefs about which state it is in, with
 10654 $q_t(s)$ indicating the degree of belief that the dialogue is in state s at time t . The POMDP
 10655 formulation can help to make dialogue systems more robust to errors, particularly in the
 10656 context of spoken language dialogues, where the speech itself may be misrecognized (Roy
 10657 et al., 2000; Williams and Young, 2007). However, finding the optimal policy in a POMDP
 10658 is computationally intractable, requiring additional approximations.

10659 19.3.3 Neural chatbots

10660 Chatting is a lot easier when you don’t need to get anything done. **Chatbots** are systems
 10661 that parry the user’s input with a response that keeps the conversation going. They can be
 10662 built from the encoder-decoder architecture discussed in § 18.3 and § 19.1.2: the encoder
 10663 converts the user’s input into a vector, and the decoder produces a sequence of words as a
 10664 response. For example, Shang et al. (2015) apply the attentional encoder-decoder transla-
 10665 tion model, training on a dataset of posts and responses from the Chinese microblogging
 10666 platform Sina Weibo.⁵ This approach is capable of generating replies that relate themati-
 10667 cally to the input, as shown in the following examples:⁶

10668 (19.10) A: High fever attacks me every New Year’s day.
 10669 B: Get well soon and stay healthy!

10670 (19.11) A: I gain one more year. Grateful to my group, so happy.
 10671 B: Getting old now. Time has no mercy.

10672 While encoder-decoder models can generate responses that make sense in the con-
 10673 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).

10674 conversations. One solution is to model the dialogue context recurrently. This creates
10675 a **hierarchical recurrent network**, including both word-level and turn-level recurrences.
10676 The turn-level hidden state is then used as additional context in the decoder (Serban et al.,
10677 2016).

10678 An open question is how to integrate the encoder-decoder architecture into task-oriented
10679 dialogue systems. Neural chatbots can be trained end-to-end: the user’s turn is analyzed
10680 by the encoder, and the system output is generated by the decoder. This architecture
10681 can be trained by log-likelihood using backpropagation (e.g., Sordoni et al., 2015; Serban
10682 et al., 2016), or by more elaborate objectives, using reinforcement learning (Li et al., 2016).
10683 In contrast, the task-oriented dialogue systems described in § 19.3.1 typically involve a
10684 set of specialized modules: one for recognizing the user input, another for deciding what
10685 action to take, and a third for arranging the text of the system output.

10686 Recurrent neural network decoders can be integrated into Markov Decision Process
10687 dialogue systems, by conditioning the decoder on a representation of the information
10688 that is to be expressed in each turn (Wen et al., 2015). Specifically, the long short-term
10689 memory (LSTM; § 6.3) architecture is augmented so that the memory cell at turn m takes
10690 an additional input d_m , which is a representation of the slots and values to be expressed
10691 in the next turn. However, this approach still relies on additional modules to recognize
10692 the user’s utterance and to plan the overall arc of the dialogue.

10693 Another promising direction is to create embeddings for the elements in the domain:
10694 for example, the slots in a record and the entities that can fill them. The encoder then
10695 encodes not only the words of the user’s input, but the embeddings of the elements that
10696 the user mentions. Similarly, the decoder is endowed with the ability to refer to specific
10697 elements in the knowledge base. He et al. (2017) show that such a method can learn to
10698 play a collaborative dialogue game, in which both players are given a list of entities and
10699 their properties, and the goal is to find an entity that is on both players’ lists.

10700 Further reading

10701 Gatt and Krahmer (2018) provide a comprehensive recent survey on text generation. For
10702 a book-length treatment of earlier work, see Reiter and Dale (2000). For a survey on image
10703 captioning, see Bernardi et al. (2016); for a survey of pre-neural approaches to dialogue
10704 systems, see Rieser and Lemon (2011). **Dialogue acts** were introduced in § 8.6 as a label-
10705 ing scheme for human-human dialogues; they also play a critical role in task-based dialogue
10706 systems (e.g., Allen et al., 1996). The incorporation of theoretical models of dialogue into
10707 computational systems is reviewed by Jurafsky and Martin (2009, chapter 24).

10708 While this chapter has focused on the informative dimension of text generation, an-
10709 other line of research aims to generate text with configurable stylistic properties (Walker
10710 et al., 1997; Mairesse and Walker, 2011; Ficler and Goldberg, 2017; Hu et al., 2017). This

10711 chapter also does not address the generation of creative text such as narratives (Riedl and
10712 Young, 2010), jokes (Ritchie, 2001), poems (Colton et al., 2012), and song lyrics (Gonçalo Oliveira
10713 et al., 2007).

10714 **Exercises**

- 10715 1. Find an article about a professional basketball game, with an associated “box score”
10716 of statistics. Which are the first three elements in the box score that are expressed
10717 in the article? Can you identify template-based patterns that express these elements
10718 of the record? Now find a second article about a different basketball game. Does it
10719 mention the same first three elements of the box score? Do your templates capture
10720 how these elements are expressed in the text?
- 10721 2. This exercise is to be done by a pair of students. One student should choose an article
10722 from the news or from Wikipedia, and manually perform semantic role labeling
10723 (SRL) on three short sentences or clauses. (See chapter 13 for a review of SRL.)
10724 Identify the main the semantic relation and its arguments and adjuncts. Pass this
10725 structured record — but not the original sentence — to the other student, whose
10726 job is to generate a sentence expressing the semantics. Then reverse roles, and try
10727 to regenerate three sentences from another article, based on the predicate-argument
10728 semantics.
- 10729 3. Compute the BLEU scores (see § 18.1.1) for the generated sentences in the previous
10730 problem, using the original article text as the reference.
- 10731 4. Align each token in the text of Figure 19.1 to a specific single record in the database,
10732 or to the null record \emptyset . For example, the tokens *south wind* would align to the record
10733 *wind direction: 06:00-21:00: mode=S*. How often is each token aligned
10734 to the same record as the previous token? How many transitions are there? How
10735 might a system learn to output *10 degrees* for the record *min=9*?
- 10736 5. In sentence compression and fusion, we may wish to preserve contiguous sequences
10737 of tokens (*n*-grams) and/or dependency edges. Find five short news articles with
10738 headlines. For each headline, compute the fraction of bigrams that appear in the
10739 main text of the article. Then do a manual dependency parse of the headline. For
10740 each dependency edge, count how often it appears as a dependency edge in the
10741 main text. You may use an automatic dependency parser to assist with this exercise,
10742 but check the output, and focus on UD 2.0 dependency grammar, as described in
10743 chapter 11.
- 10744 6. § 19.2.2 presents the idea of generating text from dependency trees, which requires
10745 **linearization**. Sometimes there are multiple ways that a dependency tree can be
10746 linearized. For example:

10747 (19.12) The sick kids stayed at home in bed.

10748 (19.13) The sick kids stayed in bed at home.

10749 Both sentences have an identical dependency parse: both *home* and *bed* are (oblique)
10750 dependents of *stayed*.

10751 Identify two more English dependency trees that can each be linearized in more than
10752 one way, and try to use a different pattern of variation in each tree. As usual, specify
10753 your trees in the Universal Dependencies 2 style, which is described in chapter 11.

7. In § 19.3.2, we considered a pizza delivery service. Let's simplify the problem to take-out, where it is only necessary to determine the topping and confirm the order. The state is a tuple in which the first element is *T* if the topping is specified and ? otherwise, and the second element is either YES or NO, depending on whether the order has been confirmed. The actions are TOPPING? (request information about the topping) and CONFIRM? (request confirmation). The state transition function is:

$$p(s_t | s_{t-1} = (?, \text{NO}), a = \text{TOPPING?}) = \begin{cases} 0.9, & s_t = (\text{T}, \text{NO}) \\ 0.1, & s_t = (?, \text{NO}). \end{cases} \quad [19.19]$$

$$p(s_t | s_{t-1} = (?, \text{NO}), a = \text{CONFIRM?}) = \begin{cases} 1, & s_t = (?, \text{NO}). \end{cases} \quad [19.20]$$

$$p(s_t | s_{t-1} = (\text{T}, \text{NO}), a = \text{TOPPING?}) = \begin{cases} 1, & s_t = (\text{T}, \text{NO}). \end{cases} \quad [19.21]$$

$$p(s_t | s_{t-1} = (\text{T}, \text{NO}), a = \text{CONFIRM?}) = \begin{cases} 0.9, & s_t = (\text{T}, \text{YES}) \\ 0.1, & s_t = (\text{T}, \text{NO}). \end{cases} \quad [19.22]$$

10754 Using the reward function defined in Equation 19.16, the discount $\gamma = 0.9$, and the
10755 initialization $V(s) = 0$, execute three iterations of Equation 19.17. After these three
10756 iterations, compute the optimal action in each state. You can assume that for the
10757 terminal states, $V(*, \text{YES}) = 0$, so you only need to compute the values for non-
10758 terminal states, $V(?, \text{NO})$ and $V(\text{T}, \text{NO})$.

- 10759 8. There are several toolkits that allow you to train encoder-decoder translation models
10760 “out of the box”, such as FairSeq (Gehring et al., 2017), xnmt (Neubig et al., 2018),
10761 tensor2tensor (Vaswani et al., 2018), and OpenNMT (Klein et al., 2017).⁷ Use one
10762 of these toolkits to train a chatbot dialogue system, using either the NPS dialogue
10763 corpus that comes with nltk (Forsyth and Martell, 2007), or, if you are feeling more
10764 ambitious, the Ubuntu dialogue corpus (Lowe et al., 2015).

⁷<https://github.com/facebookresearch/fairseq>; <https://github.com/neulab/xnmt>;
<https://github.com/tensorflow/tensor2tensor>; <http://opennmt.net/>

10765 **Appendix A**

10766 **Probability**

10767 Probability theory provides a way to reason about random events. The sorts of random
10768 events that are typically used to explain probability theory include coin flips, card draws,
10769 and the weather. It may seem odd to think about the choice of a word as akin to the flip of
10770 a coin, particularly if you are the type of person to choose words carefully. But random or
10771 not, language has proven to be extremely difficult to model deterministically. Probability
10772 offers a powerful tool for modeling and manipulating linguistic data.

10773 Probability can be thought of in terms of **random outcomes**: for example, a single coin
10774 flip has two possible outcomes, heads or tails. The set of possible outcomes is the **sample**
10775 **space**, and a subset of the **sample space** is an **event**. For a sequence of two coin flips,
10776 there are four possible outcomes, $\{HH, HT, TH, TT\}$, representing the ordered sequences
10777 heads-head, heads-tails, tails-heads, and tails-tails. The event of getting exactly one head
10778 includes two outcomes: $\{HT, TH\}$.

10779 Formally, a probability is a function from events to the interval between zero and one:
10780 $\Pr : \mathcal{F} \rightarrow [0, 1]$, where \mathcal{F} is the set of possible events. An event that is certain has proba-
10781 bility one; an event that is impossible has probability zero. For example, the probability
10782 of getting fewer than three heads on two coin flips is one. Each outcome is also an event
10783 (a set with exactly one element), and for two flips of a fair coin, the probability of each
10784 outcome is,

$$\Pr(\{HH\}) = \Pr(\{HT\}) = \Pr(\{TH\}) = \Pr(\{TT\}) = \frac{1}{4}. \quad [\text{A.1}]$$

10785 **A.1 Probabilities of event combinations**

10786 Because events are sets of outcomes, we can use set-theoretic operations such as comple-
10787 ment, intersection, and union to reason about the probabilities of events and their combi-
10788 nations.

10789 For any event A , there is a **complement** $\neg A$, such that:

- 10790 • The probability of the union $A \cup \neg A$ is $\Pr(A \cup \neg A) = 1$;
- 10791 • The intersection $A \cap \neg A = \emptyset$ is the empty set, and $\Pr(A \cap \neg A) = 0$.

10792 In the coin flip example, the event of obtaining a single head on two flips corresponds to
 10793 the set of outcomes $\{HT, TH\}$; the complement event includes the other two outcomes,
 10794 $\{TT, HH\}$.

10795 A.1.1 Probabilities of disjoint events

10796 When two events have an empty intersection, $A \cap B = \emptyset$, they are **disjoint**. The probabili-
 10797 ty of the union of two disjoint events is equal to the sum of their probabilities,

$$A \cap B = \emptyset \Rightarrow \Pr(A \cup B) = \Pr(A) + \Pr(B). \quad [A.2]$$

10798 This is the **third axiom of probability**, and it can be generalized to any countable sequence
 10799 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,

$$\Pr(\{HT, TH\}) = \Pr(\{HT\} \cup \{TH\}) = \Pr(\{HT\}) + \Pr(\{TH\}) \quad [A.3]$$

$$= \frac{1}{4} + \frac{1}{4} = \frac{1}{2}. \quad [A.4]$$

10800 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:

$$\Pr(B - (A \cap B)) = \Pr(B) - \Pr(A \cap B) \quad [A.8]$$

$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B). \quad [A.9]$$

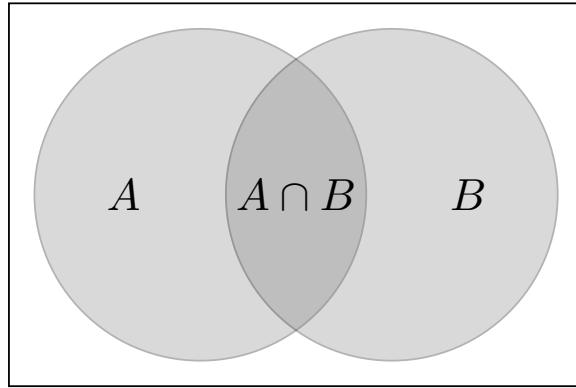


Figure A.1: A visualization of the probability of non-disjoint events A and B .

10801 A.1.2 Law of total probability

10802 A set of events $\mathcal{B} = \{B_1, B_2, \dots, B_N\}$ is a **partition** of the sample space iff each pair of
 10803 events is disjoint ($B_i \cap B_j = \emptyset$), and the union of the events is the entire sample space.
 10804 The law of total probability states that we can **marginalize** over these events as follows,

$$\Pr(A) = \sum_{B_n \in \mathcal{B}} \Pr(A \cap B_n). \quad [\text{A.10}]$$

10805 For any event B , the union $B \cup \neg B$ is a partition of the sample space. Therefore, a special
 10806 case of the law of total probability is,

$$\Pr(A) = \Pr(A \cap B) + \Pr(A \cap \neg B). \quad [\text{A.11}]$$

10807 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

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:

$$\Pr(A | B) = \frac{\Pr(A \cap B)}{\Pr(B)} = \frac{\Pr(B | A) \times \Pr(A)}{\Pr(B)} \quad [\text{A.13}]$$

10808 Each term in Bayes rule has a name, which we will occasionally use:

- 10809 • Pr(A) is the **prior**, since it is the probability of event A without knowledge about
10810 whether B happens or not.
- 10811 • Pr($B | A$) is the **likelihood**, the probability of event B given that event A has oc-
10812 curred.
- 10813 • Pr($A | B$) is the **posterior**, the probability of event A with knowledge that B has
10814 occurred.

10815 **Example** The classic examples for Bayes' rule involve tests for rare diseases, but Man-
10816 ning and Schütze (1999) reframe this example in a linguistic setting. Suppose that you are
10817 interested in a rare syntactic construction, such as *parasitic gaps*, which occur on average
10818 once in 100,000 sentences. Here is an example of a parasitic gap:

10819 (A.1) *Which class did you attend ... without registering for ...?*

10820 Lana Linguist has developed a complicated pattern matcher that attempts to identify
10821 sentences with parasitic gaps. It's pretty good, but it's not perfect:

- 10822 • If a sentence has a parasitic gap, the pattern matcher will find it with probability
10823 0.95. (This is the **recall**, which is one minus the **false positive rate**.)
- 10824 • If the sentence doesn't have a parasitic gap, the pattern matcher will wrongly say it
10825 does with probability 0.005. (This is the **false positive rate**, which is one minus the
10826 **precision**.)

10827 Suppose that Lana's pattern matcher says that a sentence contains a parasitic gap. What
10828 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:

$$\Pr(G | T) = \frac{\Pr(T | G) \times \Pr(G)}{\Pr(T)}. \quad [\text{A.14}]$$

10829 We already know both terms in the numerator: $\Pr(T | G)$ is the recall, which is 0.95; $\Pr(G)$
10830 is the prior, which is 10^{-5} .

10831 We are not given the denominator, but it can be computed using tools developed earlier
10832 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,

$$\Pr(G | T) = \frac{\Pr(T | G) \Pr(G)}{\Pr(T)} \quad [\text{A.20}]$$

$$= \frac{0.95 \times 10^{-5}}{0.95 \times 10^{-5} + 0.005 \times (1 - 10^{-5})} \quad [\text{A.21}]$$

$$\approx 0.002. \quad [\text{A.22}]$$

10833 Lana's pattern matcher seems accurate, with false positive and false negative rates
10834 below 5%. Yet the extreme rarity of the phenomenon means that a positive result from the
10835 detector is most likely to be wrong.

10836 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 function 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,

$$E[\text{price}(x)] = \int_{\mathcal{X}} \min(1, 1+x) \times \text{original-price} \times p(x)dx. \quad [A.31]$$

10871 **A.6 Modeling and estimation**

10872 **Probabilistic models** provide a principled way to reason about random events and ran-
10873 dom variables. Let's consider the coin toss example. Each toss can be modeled as a ran-
10874 dom event, with probability θ of the event H , and probability $1 - \theta$ of the complementary
10875 event T . If we write a random variable X as the total number of heads on three coin
10876 flips, then the distribution of X depends on θ . In this case, X is distributed as a **binomial**
10877 **random variable**, meaning that it is drawn from a binomial distribution, with **parameters**
10878 $(\theta, N = 3)$. This is written,

$$X \sim \text{Binomial}(\theta, N = 3). \quad [\text{A.32}]$$

10879 The properties of the binomial distribution enable us to make statements about the X ,
10880 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:

$$\hat{\theta} = \operatorname{argmax}_{\theta} p_X(x; \theta, N). \quad [\text{A.33}]$$

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,

$$p_X(x; \theta, N) = \frac{N!}{x!(N-x)!} \theta^x (1-\theta)^{N-x}. \quad [\text{A.34}]$$

10881 This likelihood is proportional to the product of the probability of individual out-
10882 comes: for example, the sequence T, H, H, T, H would have probability $\theta^3(1-\theta)^2$. The
10883 term $\frac{N!}{x!(N-x)!}$ arises from the many possible orderings by which we could obtain x heads
10884 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.

$$\ell(\theta) = x \log \theta + (N - x) \log(1 - \theta) \quad [\text{A.35}]$$

$$\frac{\partial \ell(\theta)}{\partial \theta} = \frac{x}{\theta} - \frac{N - x}{1 - \theta} \quad [\text{A.36}]$$

$$\frac{N - x}{1 - \theta} = \frac{x}{\theta} \quad [\text{A.37}]$$

$$\frac{N - x}{x} = \frac{1 - \theta}{\theta} \quad [\text{A.38}]$$

$$\frac{N}{x} - 1 = \frac{1}{\theta} - 1 \quad [\text{A.39}]$$

$$\hat{\theta} = \frac{x}{N}. \quad [\text{A.40}]$$

10885 In this case, the maximum likelihood estimate is equal to $\frac{x}{N}$, the fraction of trials that
 10886 came up heads. This intuitive solution is also known as the **relative frequency estimate**,
 10887 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:

$$p(\theta | x; N) = \frac{p(x | \theta) \times p(\theta)}{p(x)} \quad [\text{A.41}]$$

$$\propto p(x | \theta) \times p(\theta) \quad [\text{A.42}]$$

$$\hat{\theta} = \operatorname{argmax}_{\theta} p(x | \theta) \times p(\theta). \quad [\text{A.43}]$$

10888 This is the **maximum a posteriori** (MAP) estimate. Given a form for $p(\theta)$, you can de-
 10889 rive the MAP estimate using the same approach that was used to derive the maximum
 10890 likelihood estimate.

10891 Additional resources

10892 A good introduction to probability theory is offered by Manning and Schütze (1999),
 10893 which helped to motivate this section. For more detail, Sharon Goldwater provides an-
 10894 other useful reference, <http://homepages.inf.ed.ac.uk/sgwater/teaching/general/probability.pdf>. A historical and philosophical perspective on probability is offered
 10895 by Diaconis and Skyrms (2017).

10897 **Appendix B**

10898 **Numerical optimization**

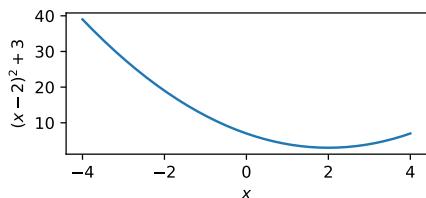
10899 Unconstrained numerical optimization involves solving problems of the form,

$$\min_{\mathbf{x} \in \mathbb{R}^D} f(\mathbf{x}), \quad [\text{B.1}]$$

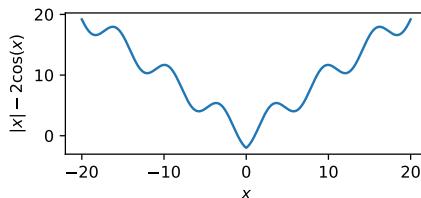
10900 where $\mathbf{x} \in \mathbb{R}^D$ is a vector of D real numbers.

10901 Differentiation is fundamental to continuous optimization. Suppose that at some \mathbf{x}^* ,
10902 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**
10903 **point** of f . For a **convex** function f (defined in § 2.3), $f(\mathbf{x}^*)$ is equal to the global minimum
10904 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,

$$\frac{\partial d}{\partial x_1} = x_1 - 2 \quad [B.2]$$

$$\frac{\partial d}{\partial x_2} = x_2 - 1 \quad [B.3]$$

10905 The unique minimum is $\mathbf{x}^* = [2, 1]^\top$.

10906 For non-convex functions, critical points are not necessarily global minima. A **local**
 10907 **minimum** \mathbf{x}^* is a point at which the function takes a smaller value than at all nearby
 10908 neighbors: formally, \mathbf{x}^* is a local minimum if there is some positive ϵ such that $f(\mathbf{x}^*) \leq$
 10909 $f(\mathbf{x})$ for all \mathbf{x} within distance ϵ of \mathbf{x}^* . Figure B.1b shows the function $f(x) = |x| - 2 \cos(x)$,
 10910 which has many local minima, as well as a unique global minimum at $x = 0$. A critical
 10911 point may also be the local or global maximum of the function; it may be a **saddle point**,
 10912 which is a minimum with respect to at least one coordinate, and a maximum with respect
 10913 to at least one other coordinate; it may be an **inflection point**, which is neither a minimum
 10914 nor maximum. When available, the second derivative of f can help to distinguish these
 10915 cases.

10916 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,

$$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} - \eta^{(t)} \nabla_{\mathbf{x}^{(t)}} f. \quad [B.4]$$

10917 where $\eta^{(t)} > 0$ is a **step size**. If the step size is chosen appropriately, this procedure will
 10918 find the global minimum of a differentiable convex function. For non-convex functions,
 10919 gradient descent will find a local minimum. The extension to non-differentiable convex
 10920 functions is discussed in § 2.3.

10921 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.

10939

Bibliography

- 10940 Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis,
10941 J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia,
10942 R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore,
10943 D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A.
10944 Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Watten-
10945 berg, M. Wicke, Y. Yu, and X. Zheng (2016). Tensorflow: Large-scale machine learning
10946 on heterogeneous distributed systems. *CoRR abs/1603.04467*.
- 10947 Abend, O. and A. Rappoport (2017). The state of the art in semantic representation. In
10948 *Proceedings of the Association for Computational Linguistics (ACL)*.
- 10949 Abney, S., R. E. Schapire, and Y. Singer (1999). Boosting applied to tagging and PP attach-
10950 ment. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.
10951 132–134.
- 10952 Abney, S. P. (1987). *The English noun phrase in its sentential aspect*. Ph. D. thesis, Mas-
10953 sachusetts Institute of Technology.
- 10954 Abney, S. P. and M. Johnson (1991). Memory requirements and local ambiguities of pars-
10955 ing strategies. *Journal of Psycholinguistic Research* 20(3), 233–250.
- 10956 Adafre, S. F. and M. De Rijke (2006). Finding similar sentences across multiple languages
10957 in wikipedia. In *Proceedings of the Workshop on NEW TEXT Wikis and blogs and other*
10958 *dynamic text sources*.
- 10959 Ahn, D. (2006). The stages of event extraction. In *Proceedings of the Workshop on Annotating*
10960 *and Reasoning about Time and Events*, pp. 1–8. Association for Computational Linguistics.
- 10961 Aho, A. V., M. S. Lam, R. Sethi, and J. D. Ullman (2006). Compilers: Principles, techniques,
10962 & tools.
- 10963 Aikhenvald, A. Y. (2004). *Evidentiality*. Oxford University Press.

- 10964 Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on
10965 Automatic Control* 19(6), 716–723.
- 10966 Akmajian, A., R. A. Demers, A. K. Farmer, and R. M. Harnish (2010). *Linguistics: An
10967 introduction to language and communication* (Sixth ed.). Cambridge, MA: MIT press.
- 10968 Alfau, F. (1999). *Chromos*. Dalkey Archive Press.
- 10969 Allauzen, C., M. Riley, J. Schalkwyk, W. Skut, and M. Mohri (2007). OpenFst: A gen-
10970 eral and efficient weighted finite-state transducer library. In *International Conference on
10971 Implementation and Application of Automata*, pp. 11–23. Springer.
- 10972 Allen, J. F. (1984). Towards a general theory of action and time. *Artificial intelligence* 23(2),
10973 123–154.
- 10974 Allen, J. F., B. W. Miller, E. K. Ringger, and T. Sikorski (1996). A robust system for natural
10975 spoken dialogue. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
10976 62–70.
- 10977 Allen, J. F., L. K. Schubert, G. Ferguson, P. Heeman, C. H. Hwang, T. Kato, M. Light,
10978 N. Martin, B. Miller, M. Poesio, and D. Traum (1995). The TRAINS project: A case
10979 study in building a conversational planning agent. *Journal of Experimental & Theoretical
10980 Artificial Intelligence* 7(1), 7–48.
- 10981 Alm, C. O., D. Roth, and R. Sproat (2005). Emotions from text: machine learning for
10982 text-based emotion prediction. In *Proceedings of Empirical Methods for Natural Language
10983 Processing (EMNLP)*, pp. 579–586.
- 10984 Aluísio, S., J. Pelizzoni, A. Marchi, L. de Oliveira, R. Manenti, and V. Marquiafável (2003).
10985 An account of the challenge of tagging a reference corpus for Brazilian Portuguese.
10986 *Computational Processing of the Portuguese Language*, 194–194.
- 10987 Anand, P., M. Walker, R. Abbott, J. E. Fox Tree, R. Bowman, and M. Minor (2011). Cats rule
10988 and dogs drool!: Classifying stance in online debate. In *Proceedings of the 2nd Workshop
10989 on Computational Approaches to Subjectivity and Sentiment Analysis*, Portland, Oregon, pp.
10990 1–9. Association for Computational Linguistics.
- 10991 Anandkumar, A. and R. Ge (2016). Efficient approaches for escaping higher order saddle
10992 points in non-convex optimization. In *Proceedings of the Conference On Learning Theory
10993 (COLT)*, pp. 81–102.
- 10994 Anandkumar, A., R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky (2014). Tensor decompo-
10995 sitions for learning latent variable models. *The Journal of Machine Learning Research* 15(1),
10996 2773–2832.

- 10997 Ando, R. K. and T. Zhang (2005). A framework for learning predictive structures from
10998 multiple tasks and unlabeled data. *The Journal of Machine Learning Research* 6, 1817–
10999 1853.
- 11000 Andor, D., C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and
11001 M. Collins (2016). Globally normalized transition-based neural networks. In *Proceedings*
11002 of the Association for Computational Linguistics (ACL), pp. 2442–2452.
- 11003 Angeli, G., P. Liang, and D. Klein (2010). A simple domain-independent probabilistic ap-
11004 proach to generation. In *Proceedings of Empirical Methods for Natural Language Processing*
11005 (EMNLP), pp. 502–512.
- 11006 Antol, S., A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh
11007 (2015). Vqa: Visual question answering. In *Proceedings of the International Conference on*
11008 *Computer Vision (ICCV)*, pp. 2425–2433.
- 11009 Aronoff, M. (1976). *Word formation in generative grammar*. MIT Press.
- 11010 Arora, S. and B. Barak (2009). *Computational complexity: a modern approach*. Cambridge
11011 University Press.
- 11012 Arora, S., R. Ge, Y. Halpern, D. Mimmo, A. Moitra, D. Sontag, Y. Wu, and M. Zhu (2013).
11013 A practical algorithm for topic modeling with provable guarantees. In *Proceedings of the*
11014 *International Conference on Machine Learning (ICML)*, pp. 280–288.
- 11015 Arora, S., Y. Li, Y. Liang, T. Ma, and A. Risteski (2016). Linear algebraic structure of word
11016 senses, with applications to polysemy. *arXiv preprint arXiv:1601.03764*.
- 11017 Artstein, R. and M. Poesio (2008). Inter-coder agreement for computational linguistics.
11018 *Computational Linguistics* 34(4), 555–596.
- 11019 Artzi, Y. and L. Zettlemoyer (2013). Weakly supervised learning of semantic parsers for
11020 mapping instructions to actions. *Transactions of the Association for Computational Linguis-*
11021 *tics* 1, 49–62.
- 11022 Attardi, G. (2006). Experiments with a multilanguage non-projective dependency parser.
11023 In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 166–170.
- 11024 Auer, P. (2013). *Code-switching in conversation: Language, interaction and identity*. Routledge.
- 11025 Auer, S., C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives (2007). Dbpedia: A
11026 nucleus for a web of open data. *The semantic web*, 722–735.
- 11027 Austin, J. L. (1962). *How to do things with words*. Oxford University Press.

- 11028 Aw, A., M. Zhang, J. Xiao, and J. Su (2006). A phrase-based statistical model for SMS text
 11029 normalization. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
 11030 33–40.
- 11031 Ba, J. L., J. R. Kiros, and G. E. Hinton (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.
- 11033 Bagga, A. and B. Baldwin (1998a). Algorithms for scoring coreference chains. In *Proceedings of the Language Resources and Evaluation Conference*, pp. 563–566.
- 11035 Bagga, A. and B. Baldwin (1998b). Entity-based cross-document coreferencing using the
 11036 vector space model. In *Proceedings of the International Conference on Computational Lin-
 11037 guistics (COLING)*, pp. 79–85.
- 11038 Bahdanau, D., K. Cho, and Y. Bengio (2014). Neural machine translation by jointly learn-
 11039 ing to align and translate. In *Neural Information Processing Systems (NIPS)*.
- 11040 Baldwin, T. and S. N. Kim (2010). Multiword expressions. In *Handbook of natural language
 11041 processing*, Volume 2, pp. 267–292. Boca Raton, USA: CRC Press.
- 11042 Balle, B., A. Quattoni, and X. Carreras (2011). A spectral learning algorithm for finite state
 11043 transducers. In *Proceedings of the European Conference on Machine Learning and Principles
 11044 and Practice of Knowledge Discovery in Databases (ECML)*, pp. 156–171.
- 11045 Banarescu, L., C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight,
 11046 P. Koehn, M. Palmer, and N. Schneider (2013, August). Abstract meaning represen-
 11047 tation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability
 11048 with Discourse*, Sofia, Bulgaria, pp. 178–186. Association for Computational
 11049 Linguistics.
- 11050 Banko, M., M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni (2007). Open
 11051 information extraction from the web. In *Proceedings of the International Joint Conference
 11052 on Artificial Intelligence (IJCAI)*, pp. 2670–2676.
- 11053 Bansal, N., A. Blum, and S. Chawla (2004). Correlation clustering. *Machine Learning* 56(1–
 11054 3), 89–113.
- 11055 Barber, D. (2012). *Bayesian reasoning and machine learning*. Cambridge University Press.
- 11056 Barman, U., A. Das, J. Wagner, and J. Foster (2014, October). Code mixing: A challenge for
 11057 language identification in the language of social media. In *Proceedings of the First Work-
 11058 shop on Computational Approaches to Code Switching*, Doha, Qatar, pp. 13–23. Association
 11059 for Computational Linguistics.

- 11060 Barnickel, T., J. Weston, R. Collobert, H.-W. Mewes, and V. Stümpflen (2009). Large scale
11061 application of neural network based semantic role labeling for automated relation ex-
11062 traction from biomedical texts. *PLoS One* 4(7), e6393.
- 11063 Baron, A. and P. Rayson (2008). Vard2: A tool for dealing with spelling variation in his-
11064 torical corpora. In *Postgraduate conference in corpus linguistics*.
- 11065 Baroni, M., R. Bernardi, and R. Zamparelli (2014). Frege in space: A program for compo-
11066 sitional distributional semantics. *Linguistic Issues in Language Technologies*.
- 11067 Barzilay, R. and M. Lapata (2008, mar). Modeling local coherence: An Entity-Based ap-
11068 proach. *Computational Linguistics* 34(1), 1–34.
- 11069 Barzilay, R. and K. R. McKeown (2005). Sentence fusion for multidocument news summa-
11070 rization. *Computational Linguistics* 31(3), 297–328.
- 11071 Beesley, K. R. and L. Karttunen (2003). *Finite-state morphology*. Stanford, CA: Center for
11072 the Study of Language and Information.
- 11073 Bejan, C. A. and S. Harabagiu (2014). Unsupervised event coreference resolution. *Compu-
11074 tational Linguistics* 40(2), 311–347.
- 11075 Bell, E. T. (1934). Exponential numbers. *The American Mathematical Monthly* 41(7), 411–419.
- 11076 Bender, E. M. (2013, jun). *Linguistic Fundamentals for Natural Language Processing: 100
11077 Essentials from Morphology and Syntax*, Volume 6 of *Synthesis Lectures on Human Language
11078 Technologies*. Morgan & Claypool Publishers.
- 11079 Bengio, S., O. Vinyals, N. Jaitly, and N. Shazeer (2015). Scheduled sampling for sequence
11080 prediction with recurrent neural networks. In *Neural Information Processing Systems
11081 (NIPS)*, pp. 1171–1179.
- 11082 Bengio, Y., R. Ducharme, P. Vincent, and C. Janvin (2003). A neural probabilistic language
11083 model. *The Journal of Machine Learning Research* 3, 1137–1155.
- 11084 Bengio, Y., P. Simard, and P. Frasconi (1994). Learning long-term dependencies with gra-
11085 dient descent is difficult. *IEEE Transactions on Neural Networks* 5(2), 157–166.
- 11086 Bengtsson, E. and D. Roth (2008). Understanding the value of features for coreference
11087 resolution. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
11088 pp. 294–303.
- 11089 Benjamini, Y. and Y. Hochberg (1995). Controlling the false discovery rate: a practical and
11090 powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B
11091 (Methodological)*, 289–300.

- 11092 Berant, J., A. Chou, R. Frostig, and P. Liang (2013). Semantic parsing on freebase from
 11093 question-answer pairs. In *Proceedings of Empirical Methods for Natural Language Processing*
 11094 (*EMNLP*), pp. 1533–1544.
- 11095 Berant, J., V. Srikumar, P.-C. Chen, A. Vander Linden, B. Harding, B. Huang, P. Clark, and
 11096 C. D. Manning (2014). Modeling biological processes for reading comprehension. In
 11097 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11098 Berg-Kirkpatrick, T., A. Bouchard-Côté, J. DeNero, and D. Klein (2010). Painless unsuper-
 11099 vised learning with features. In *Proceedings of the North American Chapter of the Associa-*
 11100 *tion for Computational Linguistics (NAACL)*, pp. 582–590.
- 11101 Berg-Kirkpatrick, T., D. Burkett, and D. Klein (2012). An empirical investigation of sta-
 11102 tistical significance in NLP. In *Proceedings of Empirical Methods for Natural Language*
 11103 *Processing (EMNLP)*, pp. 995–1005.
- 11104 Berger, A. L., V. J. D. Pietra, and S. A. D. Pietra (1996). A maximum entropy approach to
 11105 natural language processing. *Computational linguistics* 22(1), 39–71.
- 11106 Bergsma, S., D. Lin, and R. Goebel (2008). Distributional identification of non-referential
 11107 pronouns. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 10–18.
- 11108 Bernardi, R., R. Cakici, D. Elliott, A. Erdem, E. Erdem, N. Ikizler-Cinbis, F. Keller, A. Mus-
 11109 cat, and B. Plank (2016). Automatic description generation from images: A survey of
 11110 models, datasets, and evaluation measures. *Journal of Artificial Intelligence Research* 55,
 11111 409–442.
- 11112 Bertsekas, D. P. (2012). Incremental gradient, subgradient, and proximal methods for
 11113 convex optimization: A survey. See Sra et al. (2012).
- 11114 Bhatia, P., R. Guthrie, and J. Eisenstein (2016). Morphological priors for probabilistic neu-
 11115 ral word embeddings. In *Proceedings of Empirical Methods for Natural Language Processing*
 11116 (*EMNLP*).
- 11117 Bhatia, P., Y. Ji, and J. Eisenstein (2015). Better document-level sentiment analysis from
 11118 first discourse parsing. In *Proceedings of Empirical Methods for Natural Language Processing*
 11119 (*EMNLP*).
- 11120 Biber, D. (1991). *Variation across speech and writing*. Cambridge University Press.
- 11121 Bird, S., E. Klein, and E. Loper (2009). *Natural language processing with Python*. California:
 11122 O'Reilly Media.
- 11123 Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

- 11124 Björkelund, A. and P. Nugues (2011). Exploring lexicalized features for coreference reso-
11125 lution. In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 45–50.
- 11126 Blackburn, P. and J. Bos (2005). *Representation and inference for natural language: A first*
11127 *course in computational semantics*. CSLI.
- 11128 Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM* 55(4), 77–84.
- 11129 Blei, D. M. (2014). Build, compute, critique, repeat: Data analysis with latent variable
11130 models. *Annual Review of Statistics and Its Application* 1, 203–232.
- 11131 Blei, D. M., A. Y. Ng, and M. I. Jordan (2003). Latent dirichlet allocation. *the Journal of*
11132 *machine Learning research* 3, 993–1022.
- 11133 Blitzer, J., M. Dredze, and F. Pereira (2007). Biographies, bollywood, boom-boxes and
11134 blenders: Domain adaptation for sentiment classification. In *Proceedings of the Associa-*
11135 *tion for Computational Linguistics (ACL)*, pp. 440–447.
- 11136 Blum, A. and T. Mitchell (1998). Combining labeled and unlabeled data with co-training.
11137 In *Proceedings of the Conference On Learning Theory (COLT)*, pp. 92–100.
- 11138 Bobrow, D. G., R. M. Kaplan, M. Kay, D. A. Norman, H. Thompson, and T. Winograd
11139 (1977). Gus, a frame-driven dialog system. *Artificial intelligence* 8(2), 155–173.
- 11140 Bochnet, B. (2010). Very high accuracy and fast dependency parsing is not a contradiction.
11141 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.
11142 89–97.
- 11143 Boitet, C. (1988). Pros and cons of the pivot and transfer approaches in multilingual ma-
11144 chine translation. *Readings in machine translation*, 273–279.
- 11145 Bojanowski, P., E. Grave, A. Joulin, and T. Mikolov (2017). Enriching word vectors with
11146 subword information. *Transactions of the Association for Computational Linguistics* 5, 135–
11147 146.
- 11148 Bollacker, K., C. Evans, P. Paritosh, T. Sturge, and J. Taylor (2008). Freebase: a collabora-
11149 tively created graph database for structuring human knowledge. In *Proceedings of the*
11150 *ACM International Conference on Management of Data (SIGMOD)*, pp. 1247–1250. AcM.
- 11151 Bolukbasi, T., K.-W. Chang, J. Y. Zou, V. Saligrama, and A. T. Kalai (2016). Man is to
11152 computer programmer as woman is to homemaker? debiasing word embeddings. In *Neural*
11153 *Information Processing Systems (NIPS)*, pp. 4349–4357.
- 11154 Bordes, A., N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko (2013). Translating
11155 embeddings for modeling multi-relational data. In *Neural Information Processing Systems*
11156 (*NIPS*), pp. 2787–2795.

- 11157 Bordes, A., J. Weston, R. Collobert, Y. Bengio, et al. (2011). Learning structured embed-
11158 dings of knowledge bases. In *Proceedings of the National Conference on Artificial Intelligence
11159 (AAAI)*, pp. 301–306.
- 11160 Borges, J. L. (1993). *Other Inquisitions 1937–1952*. University of Texas Press. Translated by
11161 Ruth L. C. Simms.
- 11162 Botha, J. A. and P. Blunsom (2014). Compositional morphology for word representations
11163 and language modelling. In *Proceedings of the International Conference on Machine Learn-
11164 ing (ICML)*.
- 11165 Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*,
11166 pp. 421–436. Springer.
- 11167 Bottou, L., F. E. Curtis, and J. Nocedal (2016). Optimization methods for large-scale ma-
11168 chine learning. *arXiv preprint arXiv:1606.04838*.
- 11169 Bowman, S. R., L. Vilnis, O. Vinyals, A. Dai, R. Jozefowicz, and S. Bengio (2016). Gen-
11170 erating sentences from a continuous space. In *Proceedings of the Conference on Natural
11171 Language Learning (CoNLL)*, pp. 10–21.
- 11172 boyd, d. and K. Crawford (2012). Critical questions for big data. *Information, Communica-
11173 tion & Society* 15(5), 662–679.
- 11174 Boyd, S. and L. Vandenberghe (2004). *Convex Optimization*. New York: Cambridge Uni-
11175 versity Press.
- 11176 Branavan, S., H. Chen, J. Eisenstein, and R. Barzilay (2009). Learning document-level
11177 semantic properties from free-text annotations. *Journal of Artificial Intelligence Re-
11178 search* 34(2), 569–603.
- 11179 Branavan, S. R., H. Chen, L. S. Zettlemoyer, and R. Barzilay (2009). Reinforcement learning
11180 for mapping instructions to actions. In *Proceedings of the Association for Computational
11181 Linguistics (ACL)*, pp. 82–90.
- 11182 BRANTS, T. and A. Franz (2006). The google 1t 5-gram corpus. LDC2006T13.
- 11183 Braud, C., O. Lacroix, and A. Søgaard (2017). Does syntax help discourse segmenta-
11184 tion? not so much. In *Proceedings of Empirical Methods for Natural Language Processing
11185 (EMNLP)*, pp. 2432–2442.
- 11186 Briscoe, T. (2011). Introduction to formal semantics for natural language.
- 11187 Brown, P. F., J. Cocke, S. A. D. Pietra, V. J. D. Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer,
11188 and P. S. Roossin (1990). A statistical approach to machine translation. *Computational
11189 linguistics* 16(2), 79–85.

- 11190 Brown, P. F., P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai (1992). Class-based
11191 n-gram models of natural language. *Computational linguistics* 18(4), 467–479.
- 11192 Brown, P. F., V. J. D. Pietra, S. A. D. Pietra, and R. L. Mercer (1993). The mathematics
11193 of statistical machine translation: Parameter estimation. *Computational linguistics* 19(2),
11194 263–311.
- 11195 Brun, C. and C. Roux (2014). Décomposition des “hash tags” pour l’amélioration de la
11196 classification en polarité des “tweets”. *Proceedings of Traitement Automatique des Langues
11197 Naturelles*, 473–478.
- 11198 Bruni, E., N.-K. Tran, and M. Baroni (2014). Multimodal distributional semantics. *Journal
11199 of Artificial Intelligence Research* 49(2014), 1–47.
- 11200 Bullinaria, J. A. and J. P. Levy (2007). Extracting semantic representations from word co-
11201 occurrence statistics: A computational study. *Behavior research methods* 39(3), 510–526.
- 11202 Bunescu, R. C. and R. J. Mooney (2005). A shortest path dependency kernel for relation
11203 extraction. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
11204 pp. 724–731.
- 11205 Bunescu, R. C. and M. Pasca (2006). Using encyclopedic knowledge for named entity
11206 disambiguation. In *Proceedings of the European Chapter of the Association for Computational
11207 Linguistics (EACL)*, pp. 9–16.
- 11208 Burstein, J., D. Marcu, and K. Knight (2003). Finding the WRITE stuff: Automatic identi-
11209 fication of discourse structure in student essays. *IEEE Intelligent Systems* 18(1), 32–39.
- 11210 Burstein, J., J. Tetreault, and S. Andreyev (2010). Using entity-based features to model
11211 coherence in student essays. In *Human language technologies: The 2010 annual conference
11212 of the North American chapter of the Association for Computational Linguistics*, pp. 681–684.
11213 Association for Computational Linguistics.
- 11214 Burstein, J., J. Tetreault, and M. Chodorow (2013). Holistic discourse coherence annotation
11215 for noisy essay writing. *Dialogue & Discourse* 4(2), 34–52.
- 11216 Cai, Q. and A. Yates (2013). Large-scale semantic parsing via schema matching and lexicon
11217 extension. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 423–
11218 433.
- 11219 Caliskan, A., J. J. Bryson, and A. Narayanan (2017). Semantics derived automatically from
11220 language corpora contain human-like biases. *Science* 356(6334), 183–186.
- 11221 Canny, J. (1987). A computational approach to edge detection. In *Readings in Computer
11222 Vision*, pp. 184–203. Elsevier.

- 11223 Cappé, O. and E. Moulines (2009). On-line expectation–maximization algorithm for latent
 11224 data models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 71(3),
 11225 593–613.
- 11226 Carbonell, J. and J. Goldstein (1998). The use of mmr, diversity-based reranking for re-
 11227 ordering documents and producing summaries. In *Proceedings of ACM SIGIR conference*
 11228 on Research and development in information retrieval, pp. 335–336.
- 11229 Carbonell, J. R. (1970). Mixed-initiative man-computer instructional dialogues. Technical
 11230 report, BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS.
- 11231 Cardie, C. and K. Wagstaff (1999). Noun phrase coreference as clustering. In *Proceedings*
 11232 of Empirical Methods for Natural Language Processing (EMNLP), pp. 82–89.
- 11233 Carletta, J. (1996). Assessing agreement on classification tasks: the kappa statistic. *Com-
 11234 putational linguistics* 22(2), 249–254.
- 11235 Carletta, J. (2007). Unleashing the killer corpus: experiences in creating the multi-
 11236 everything ami meeting corpus. *Language Resources and Evaluation* 41(2), 181–190.
- 11237 Carlson, L. and D. Marcu (2001). Discourse tagging reference manual. Technical Report
 11238 ISI-TR-545, Information Sciences Institute.
- 11239 Carlson, L., M. E. Okurowski, and D. Marcu (2002). RST discourse treebank. Linguistic
 11240 Data Consortium, University of Pennsylvania.
- 11241 Carpenter, B. (1997). *Type-logical semantics*. Cambridge, MA: MIT Press.
- 11242 Carreras, X., M. Collins, and T. Koo (2008). Tag, dynamic programming, and the percep-
 11243 tron for efficient, feature-rich parsing. In *Proceedings of the Conference on Natural Language*
 11244 *Learning (CoNLL)*, pp. 9–16.
- 11245 Carreras, X. and L. Màrquez (2005). Introduction to the conll-2005 shared task: Semantic
 11246 role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language*
 11247 *Learning*, pp. 152–164. Association for Computational Linguistics.
- 11248 Carroll, L. (1865). *Alice’s Adventures in Wonderland*. London: Macmillan.
- 11249 Carroll, L. (1917). *Through the looking glass: And what Alice found there*. Chicago: Rand,
 11250 McNally.
- 11251 Chambers, N. and D. Jurafsky (2008). Jointly combining implicit constraints improves
 11252 temporal ordering. In *Proceedings of Empirical Methods for Natural Language Processing*
 11253 (EMNLP), pp. 698–706.

- 11254 Chang, K.-W., A. Krishnamurthy, A. Agarwal, H. Daume III, and J. Langford (2015).
11255 Learning to search better than your teacher. In *Proceedings of the International Conference on Machine Learning (ICML)*.
11256
- 11257 Chang, M.-W., L. Ratinov, and D. Roth (2007). Guiding semi-supervision with constraint-
11258 driven learning. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
11259 280–287.
- 11260 Chang, M.-W., L.-A. Ratinov, N. Rizzolo, and D. Roth (2008). Learning and inference with
11261 constraints. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp.
11262 1513–1518.
- 11263 Chapman, W. W., W. Bridewell, P. Hanbury, G. F. Cooper, and B. G. Buchanan (2001). A
11264 simple algorithm for identifying negated findings and diseases in discharge summaries.
11265 *Journal of biomedical informatics* 34(5), 301–310.
- 11266 Charniak, E. (1997). Statistical techniques for natural language parsing. *AI magazine* 18(4),
11267 33–43.
- 11268 Charniak, E. and M. Johnson (2005). Coarse-to-fine n-best parsing and maxent discrimi-
11269 native reranking. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
11270 173–180.
- 11271 Chelba, C. and A. Acero (2006). Adaptation of maximum entropy capitalizer: Little data
11272 can help a lot. *Computer Speech & Language* 20(4), 382–399.
- 11273 Chelba, C., T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson (2013).
11274 One billion word benchmark for measuring progress in statistical language modeling.
11275 *arXiv preprint arXiv:1312.3005*.
- 11276 Chen, D., J. Bolton, and C. D. Manning (2016). A thorough examination of the CNN/Daily
11277 Mail reading comprehension task. In *Proceedings of the Association for Computational
11278 Linguistics (ACL)*.
- 11279 Chen, D. and C. D. Manning (2014). A fast and accurate dependency parser using neural
11280 networks. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
11281 pp. 740–750.
- 11282 Chen, D. L. and R. J. Mooney (2008). Learning to sportscast: a test of grounded language
11283 acquisition. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp.
11284 128–135.
- 11285 Chen, H., S. Branavan, R. Barzilay, and D. R. Karger (2009). Content modeling using latent
11286 permutations. *Journal of Artificial Intelligence Research* 36(1), 129–163.

- 11287 Chen, M., Z. Xu, K. Weinberger, and F. Sha (2012). Marginalized denoising autoencoders
11288 for domain adaptation. In *Proceedings of the International Conference on Machine Learning*
11289 (*ICML*).
- 11290 Chen, M. X., O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones, N. Parmar,
11291 M. Schuster, Z. Chen, Y. Wu, and M. Hughes (2018). The best of both worlds: Combin-
11292 ing recent advances in neural machine translation. In *Proceedings of the Association for*
11293 *Computational Linguistics (ACL)*.
- 11294 Chen, S. F. and J. Goodman (1999). An empirical study of smoothing techniques for lan-
11295 guage modeling. *Computer Speech & Language* 13(4), 359–393.
- 11296 Chen, T. and C. Guestrin (2016). Xgboost: A scalable tree boosting system. In *Proceedings*
11297 *of Knowledge Discovery and Data Mining (KDD)*, pp. 785–794.
- 11298 Chen, X., X. Qiu, C. Zhu, P. Liu, and X. Huang (2015). Long short-term memory neural
11299 networks for chinese word segmentation. In *Proceedings of Empirical Methods for Natural*
11300 *Language Processing (EMNLP)*, pp. 1197–1206.
- 11301 Chen, Y., S. Gilroy, A. Malletti, K. Knight, and J. May (2018). Recurrent neural networks
11302 as weighted language recognizers. In *Proceedings of the North American Chapter of the*
11303 *Association for Computational Linguistics (NAACL)*.
- 11304 Chen, Z. and H. Ji (2009). Graph-based event coreference resolution. In *Proceedings of*
11305 *the 2009 Workshop on Graph-based Methods for Natural Language Processing*, pp. 54–57.
11306 Association for Computational Linguistics.
- 11307 Cheng, X. and D. Roth (2013). Relational inference for wikification. In *Proceedings of*
11308 *Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1787–1796.
- 11309 Chiang, D. (2007). Hierarchical phrase-based translation. *Computational Linguistics* 33(2),
11310 201–228.
- 11311 Chiang, D., J. Graehl, K. Knight, A. Pauls, and S. Ravi (2010). Bayesian inference for
11312 finite-state transducers. In *Proceedings of the North American Chapter of the Association for*
11313 *Computational Linguistics (NAACL)*, pp. 447–455.
- 11314 Chinchor, N. and P. Robinson (1997). Muc-7 named entity task definition. In *Proceedings*
11315 *of the 7th Conference on Message Understanding*, Volume 29.
- 11316 Cho, K. (2015). Natural language understanding with distributed representation.
11317 *CoRR abs/1511.07916*.

- 11318 Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and
11319 Y. Bengio (2014). Learning phrase representations using rnn encoder-decoder for sta-
11320 tistical machine translation. In *Proceedings of Empirical Methods for Natural Language*
11321 *Processing (EMNLP)*.
- 11322 Chomsky, N. (1957). *Syntactic structures*. The Hague: Mouton & Co.
- 11323 Chomsky, N. (1982). *Some concepts and consequences of the theory of government and binding*,
11324 Volume 6. MIT press.
- 11325 Choromanska, A., M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun (2015). The loss
11326 surfaces of multilayer networks. In *Proceedings of Artificial Intelligence and Statistics (AIS-*
11327 *TATS*), pp. 192–204.
- 11328 Christensen, J., S. Soderland, O. Etzioni, et al. (2010). Semantic role labeling for open
11329 information extraction. In *Proceedings of the Workshop on Formalisms and Methodology for*
11330 *Learning by Reading*, pp. 52–60. Association for Computational Linguistics.
- 11331 Christodoulopoulos, C., S. Goldwater, and M. Steedman (2010). Two decades of unsuper-
11332 vised pos induction: How far have we come? In *Proceedings of Empirical Methods for*
11333 *Natural Language Processing (EMNLP)*, pp. 575–584.
- 11334 Chu, Y.-J. and T.-H. Liu (1965). On shortest arborescence of a directed graph. *Scientia*
11335 *Sinica* 14(10), 1396–1400.
- 11336 Chung, C. and J. W. Pennebaker (2007). The psychological functions of function words.
11337 In K. Fiedler (Ed.), *Social communication*, pp. 343–359. New York and Hove: Psychology
11338 Press.
- 11339 Church, K. (2011). A pendulum swung too far. *Linguistic Issues in Language Technology* 6(5),
11340 1–27.
- 11341 Church, K. W. (2000). Empirical estimates of adaptation: the chance of two Noriega-
11342 s is closer to $p/2$ than p^2 . In *Proceedings of the International Conference on Computational*
11343 *Linguistics (COLING)*, pp. 180–186.
- 11344 Church, K. W. and P. Hanks (1990). Word association norms, mutual information, and
11345 lexicography. *Computational linguistics* 16(1), 22–29.
- 11346 Ciaramita, M. and M. Johnson (2003). Supersense tagging of unknown nouns in wordnet.
11347 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 168–
11348 175.
- 11349 Clark, K. and C. D. Manning (2015). Entity-centric coreference resolution with model
11350 stacking. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1405–
11351 1415.

- 11352 Clark, K. and C. D. Manning (2016). Improving coreference resolution by learning entity-
 11353 level distributed representations. In *Proceedings of the Association for Computational Lin-*
 11354 *guistics (ACL)*.
- 11355 Clark, P. (2015). Elementary school science and math tests as a driver for ai: take the aristo
 11356 challenge! In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp.
 11357 4019–4021.
- 11358 Clarke, J., D. Goldwasser, M.-W. Chang, and D. Roth (2010). Driving semantic parsing
 11359 from the world’s response. In *Proceedings of the Conference on Natural Language Learning*
 11360 (*CoNLL*), pp. 18–27.
- 11361 Clarke, J. and M. Lapata (2008). Global inference for sentence compression: An integer
 11362 linear programming approach. *Journal of Artificial Intelligence Research* 31, 399–429.
- 11363 Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and psychologi-*
 11364 *cal measurement* 20(1), 37–46.
- 11365 Cohen, S. (2016). *Bayesian analysis in natural language processing*. Synthesis Lectures on
 11366 Human Language Technologies. San Rafael, CA: Morgan & Claypool Publishers.
- 11367 Collier, N., C. Nobata, and J.-i. Tsujii (2000). Extracting the names of genes and gene
 11368 products with a hidden markov model. In *Proceedings of the International Conference on*
 11369 *Computational Linguistics (COLING)*, pp. 201–207.
- 11370 Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *Proceed-*
 11371 *ings of the Association for Computational Linguistics (ACL)*, pp. 16–23.
- 11372 Collins, M. (2002). Discriminative training methods for hidden markov models: theory
 11373 and experiments with perceptron algorithms. In *Proceedings of Empirical Methods for*
 11374 *Natural Language Processing (EMNLP)*, pp. 1–8.
- 11375 Collins, M. (2013). Notes on natural language processing. <http://www.cs.columbia.edu/~mcollins/notes-spring2013.html>.
- 11377 Collins, M. and T. Koo (2005). Discriminative reranking for natural language parsing.
 11378 *Computational Linguistics* 31(1), 25–70.
- 11379 Collins, M. and B. Roark (2004). Incremental parsing with the perceptron algorithm. In
 11380 *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pp.
 11381 111. Association for Computational Linguistics.
- 11382 Collobert, R., K. Kavukcuoglu, and C. Farabet (2011). Torch7: A matlab-like environment
 11383 for machine learning. Technical Report EPFL-CONF-192376, EPFL.

- 11384 Collobert, R. and J. Weston (2008). A unified architecture for natural language process-
11385 ing: Deep neural networks with multitask learning. In *Proceedings of the International*
11386 *Conference on Machine Learning (ICML)*, pp. 160–167.
- 11387 Collobert, R., J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa (2011). Nat-
11388 ural language processing (almost) from scratch. *Journal of Machine Learning Research* 12,
11389 2493–2537.
- 11390 Colton, S., J. Goodwin, and T. Veale (2012). Full-face poetry generation. In *Proceedings of*
11391 *the International Conference on Computational Creativity*, pp. 95–102.
- 11392 Conneau, A., D. Kiela, H. Schwenk, L. Barrault, and A. Bordes (2017). Supervised learning
11393 of universal sentence representations from natural language inference data. In *Proceed-
11394 ings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 681–691.
- 11395 Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2009). *Introduction to algorithms*
11396 (third ed.). MIT press.
- 11397 Cotterell, R., H. Schütze, and J. Eisner (2016). Morphological smoothing and extrapolation
11398 of word embeddings. In *Proceedings of the Association for Computational Linguistics (ACL)*,
11399 pp. 1651–1660.
- 11400 Coviello, L., Y. Sohn, A. D. Kramer, C. Marlow, M. Franceschetti, N. A. Christakis, and
11401 J. H. Fowler (2014). Detecting emotional contagion in massive social networks. *PloS
11402 one* 9(3), e90315.
- 11403 Covington, M. A. (2001). A fundamental algorithm for dependency parsing. In *Proceedings
11404 of the 39th annual ACM southeast conference*, pp. 95–102.
- 11405 Crammer, K., O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer (2006, December).
11406 Online passive-aggressive algorithms. *The Journal of Machine Learning Research* 7, 551–
11407 585.
- 11408 Crammer, K. and Y. Singer (2001). Pranking with ranking. In *Neural Information Processing
11409 Systems (NIPS)*, pp. 641–647.
- 11410 Creutz, M. and K. Lagus (2007). Unsupervised models for morpheme segmentation and
11411 morphology learning. *ACM Transactions on Speech and Language Processing (TSLP)* 4(1),
11412 3.
- 11413 Cross, J. and L. Huang (2016). Span-based constituency parsing with a structure-label
11414 system and provably optimal dynamic oracles. In *Proceedings of Empirical Methods for
11415 Natural Language Processing (EMNLP)*, pp. 1–11.

- 11416 Cucerzan, S. (2007). Large-scale named entity disambiguation based on wikipedia data.
 11417 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11418 Cui, H., R. Sun, K. Li, M.-Y. Kan, and T.-S. Chua (2005). Question answering passage
 11419 retrieval using dependency relations. In *Proceedings of the 28th annual international ACM*
 11420 *SIGIR conference on Research and development in information retrieval*, pp. 400–407. ACM.
- 11421 Cui, Y., Z. Chen, S. Wei, S. Wang, T. Liu, and G. Hu (2017). Attention-over-attention neural
 11422 networks for reading comprehension. In *Proceedings of the Association for Computational*
 11423 *Linguistics (ACL)*.
- 11424 Culotta, A. and J. Sorensen (2004). Dependency tree kernels for relation extraction. In
 11425 *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11426 Culotta, A., M. Wick, and A. McCallum (2007). First-order probabilistic models for coref-
 11427 erence resolution. In *Proceedings of the North American Chapter of the Association for Com-*
 11428 *putational Linguistics (NAACL)*, pp. 81–88.
- 11429 Curry, H. B. and R. Feys (1958). *Combinatory Logic*, Volume I. Amsterdam: North Holland.
- 11430 Danescu-Niculescu-Mizil, C., M. Sudhof, D. Jurafsky, J. Leskovec, and C. Potts (2013). A
 11431 computational approach to politeness with application to social factors. In *Proceedings*
 11432 *of the Association for Computational Linguistics (ACL)*, pp. 250–259.
- 11433 Das, D., D. Chen, A. F. Martins, N. Schneider, and N. A. Smith (2014). Frame-semantic
 11434 parsing. *Computational Linguistics* 40(1), 9–56.
- 11435 Daumé III, H. (2007). Frustratingly easy domain adaptation. In *Proceedings of the Associa-*
 11436 *tion for Computational Linguistics (ACL)*.
- 11437 Daumé III, H., J. Langford, and D. Marcu (2009). Search-based structured prediction.
 11438 *Machine learning* 75(3), 297–325.
- 11439 Daumé III, H. and D. Marcu (2005). A large-scale exploration of effective global features
 11440 for a joint entity detection and tracking model. In *Proceedings of Empirical Methods for*
 11441 *Natural Language Processing (EMNLP)*, pp. 97–104.
- 11442 Dauphin, Y. N., R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio (2014). Iden-
 11443 tifying and attacking the saddle point problem in high-dimensional non-convex opti-
 11444 mization. In *Neural Information Processing Systems (NIPS)*, pp. 2933–2941.
- 11445 Davidson, D. (1967). The logical form of action sentences. In N. Rescher (Ed.), *The Logic of*
 11446 *Decision and Action*. Pittsburgh: University of Pittsburgh Press.

- 11447 De Gispert, A. and J. B. Marino (2006). Catalan-english statistical machine translation
11448 without parallel corpus: bridging through spanish. In *Proc. of 5th International Conference*
11449 *on Language Resources and Evaluation (LREC)*, pp. 65–68. Citeseer.
- 11450 De Marneffe, M.-C. and C. D. Manning (2008). The stanford typed dependencies represen-
11451 tation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain*
11452 *Parser Evaluation*, pp. 1–8. Association for Computational Linguistics.
- 11453 Dean, J. and S. Ghemawat (2008). Mapreduce: simplified data processing on large clusters.
11454 *Communications of the ACM* 51(1), 107–113.
- 11455 Deerwester, S. C., S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman (1990).
11456 Indexing by latent semantic analysis. *JASIS* 41(6), 391–407.
- 11457 Dehdari, J. (2014). *A Neurophysiologically-Inspired Statistical Language Model*. Ph. D. thesis,
11458 The Ohio State University.
- 11459 Deisenroth, M. P., A. A. Faisal, and C. S. Ong (2018). *Mathematics For Machine Learning*.
11460 Cambridge UP.
- 11461 Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). Maximum likelihood from incom-
11462 plete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodo-*
11463 *logical)*, 1–38.
- 11464 Denis, P. and J. Baldridge (2007). A ranking approach to pronoun resolution. In *IJCAI*.
- 11465 Denis, P. and J. Baldridge (2008). Specialized models and ranking for coreference resolu-
11466 tion. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*,
11467 EMNLP '08, Stroudsburg, PA, USA, pp. 660–669. Association for Computational Lin-
11468 guistics.
- 11469 Denis, P. and J. Baldridge (2009). Global joint models for coreference resolution and named
11470 entity classification. *Procesamiento del Lenguaje Natural* 42.
- 11471 Derrida, J. (1985). Des tours de babel. In J. Graham (Ed.), *Difference in translation*. Ithaca,
11472 NY: Cornell University Press.
- 11473 Dhingra, B., H. Liu, Z. Yang, W. W. Cohen, and R. Salakhutdinov (2017). Gated-attention
11474 readers for text comprehension. In *Proceedings of the Association for Computational Lin-*
11475 *guistics (ACL)*.
- 11476 Diaconis, P. and B. Skyrms (2017). *Ten Great Ideas About Chance*. Princeton University
11477 Press.
- 11478 Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classifica-
11479 tion learning algorithms. *Neural computation* 10(7), 1895–1923.

- 11480 Dietterich, T. G., R. H. Lathrop, and T. Lozano-Pérez (1997). Solving the multiple instance
11481 problem with axis-parallel rectangles. *Artificial intelligence* 89(1), 31–71.
- 11482 Dimitrova, L., N. Ide, V. Petkevic, T. Erjavec, H. J. Kaalep, and D. Tufis (1998). Multext-
11483 east: Parallel and comparable corpora and lexicons for six central and eastern european
11484 languages. In *Proceedings of the 17th international conference on Computational linguistics-*
11485 *Volume 1*, pp. 315–319. Association for Computational Linguistics.
- 11486 Doddington, G. R., A. Mitchell, M. A. Przybocki, L. A. Ramshaw, S. Strassel, and R. M.
11487 Weischedel (2004). The automatic content extraction (ace) program-tasks, data, and
11488 evaluation. In *Proceedings of the Language Resources and Evaluation Conference*, pp. 837–
11489 840.
- 11490 dos Santos, C., B. Xiang, and B. Zhou (2015). Classifying relations by ranking with con-
11491 volutional neural networks. In *Proceedings of the Association for Computational Linguistics*
11492 (*ACL*), pp. 626–634.
- 11493 Dowty, D. (1991). Thematic proto-roles and argument selection. *Language*, 547–619.
- 11494 Dredze, M., P. McNamee, D. Rao, A. Gerber, and T. Finin (2010). Entity disambiguation
11495 for knowledge base population. In *Proceedings of the 23rd International Conference on*
11496 *Computational Linguistics*, pp. 277–285. Association for Computational Linguistics.
- 11497 Dredze, M., M. J. Paul, S. Bergsma, and H. Tran (2013). Carmen: A Twitter geolocation
11498 system with applications to public health. In *AAAI workshop on expanding the boundaries*
11499 *of health informatics using AI (HIAI)*, pp. 20–24.
- 11500 Dreyfus, H. L. (1992). *What computers still can't do: a critique of artificial reason*. MIT press.
- 11501 Du, L., W. Buntine, and M. Johnson (2013). Topic segmentation with a structured topic
11502 model. In *Proceedings of the North American Chapter of the Association for Computational*
11503 *Linguistics (NAACL)*, pp. 190–200.
- 11504 Duchi, J., E. Hazan, and Y. Singer (2011). Adaptive subgradient methods for online learn-
11505 ing and stochastic optimization. *The Journal of Machine Learning Research* 12, 2121–2159.
- 11506 Dunietz, J., L. Levin, and J. Carbonell (2017). The because corpus 2.0: Annotating causality
11507 and overlapping relations. In *Proceedings of the Linguistic Annotation Workshop*.
- 11508 Durrett, G., T. Berg-Kirkpatrick, and D. Klein (2016). Learning-based single-document
11509 summarization with compression and anaphoricity constraints. In *Proceedings of the*
11510 *Association for Computational Linguistics (ACL)*, pp. 1998–2008.
- 11511 Durrett, G. and D. Klein (2013). Easy victories and uphill battles in coreference resolution.
11512 In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

- 11513 Durrett, G. and D. Klein (2015). Neural crf parsing. In *Proceedings of the Association for*
11514 *Computational Linguistics (ACL)*.
- 11515 Dyer, C., M. Ballesteros, W. Ling, A. Matthews, and N. A. Smith (2015). Transition-based
11516 dependency parsing with stack long short-term memory. In *Proceedings of the Association*
11517 *for Computational Linguistics (ACL)*, pp. 334–343.
- 11518 Dyer, C., A. Kuncoro, M. Ballesteros, and N. A. Smith (2016). Recurrent neural network
11519 grammars. In *Proceedings of the North American Chapter of the Association for Computational*
11520 *Linguistics (NAACL)*, pp. 199–209.
- 11521 Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of*
11522 *Standards B* 71(4), 233–240.
- 11523 Efron, B. and R. J. Tibshirani (1993). An introduction to the bootstrap: Monographs on
11524 statistics and applied probability, vol. 57. *New York and London: Chapman and Hall/CRC*.
- 11525 Eisenstein, J. (2009). Hierarchical text segmentation from multi-scale lexical cohesion. In
11526 *Proceedings of the North American Chapter of the Association for Computational Linguistics*
11527 *(NAACL)*.
- 11528 Eisenstein, J. and R. Barzilay (2008). Bayesian unsupervised topic segmentation. In *Pro-*
11529 *ceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11530 Eisner, J. (1997). State-of-the-art algorithms for minimum spanning trees: A tutorial dis-
11531 cussion.
- 11532 Eisner, J. (2000). Bilexical grammars and their cubic-time parsing algorithms. In *Advances*
11533 *in probabilistic and other parsing technologies*, pp. 29–61. Springer.
- 11534 Eisner, J. (2002). Parameter estimation for probabilistic finite-state transducers. In *Proceed-*
11535 *ings of the Association for Computational Linguistics (ACL)*, pp. 1–8.
- 11536 Eisner, J. (2016). Inside-outside and forward-backward algorithms are just backprop. In
11537 *Proceedings of the Workshop on Structured Prediction for NLP*, pp. 1–17.
- 11538 Eisner, J. M. (1996). Three new probabilistic models for dependency parsing: An explo-
11539 ration. In *Proceedings of the International Conference on Computational Linguistics (COL-*
11540 *ING)*, pp. 340–345.
- 11541 Ekman, P. (1992). Are there basic emotions? *Psychological Review* 99(3), 550–553.
- 11542 Elman, J. L. (1990). Finding structure in time. *Cognitive science* 14(2), 179–211.

- 11543 Elman, J. L., E. A. Bates, M. H. Johnson, A. Karmiloff-Smith, D. Parisi, and K. Plunkett
11544 (1998). *Rethinking innateness: A connectionist perspective on development*, Volume 10. MIT
11545 press.
- 11546 Elsner, M. and E. Charniak (2010). Disentangling chat. *Computational Linguistics* 36(3),
11547 389–409.
- 11548 Esuli, A. and F. Sebastiani (2006). Sentiwordnet: A publicly available lexical resource for
11549 opinion mining. In *LREC*, Volume 6, pp. 417–422. Citeseer.
- 11550 Etzioni, O., A. Fader, J. Christensen, S. Soderland, and M. Mausam (2011). Open informa-
11551 tion extraction: The second generation. In *Proceedings of the International Joint Conference*
11552 on *Artificial Intelligence (IJCAI)*, pp. 3–10.
- 11553 Faruqui, M., J. Dodge, S. K. Jauhar, C. Dyer, E. Hovy, and N. A. Smith (2015). Retrofitting
11554 word vectors to semantic lexicons. In *Proceedings of the North American Chapter of the*
11555 *Association for Computational Linguistics (NAACL)*.
- 11556 Faruqui, M. and C. Dyer (2014). Improving vector space word representations using mul-
11557 tilingual correlation. In *Proceedings of the European Chapter of the Association for Compu-*
11558 *tational Linguistics (EACL)*, pp. 462–471.
- 11559 Faruqui, M., R. McDonald, and R. Soricut (2016). Morpho-syntactic lexicon generation
11560 using graph-based semi-supervised learning. *Transactions of the Association for Compu-*
11561 *tational Linguistics* 4, 1–16.
- 11562 Faruqui, M., Y. Tsvetkov, P. Rastogi, and C. Dyer (2016, August). Problems with evaluation
11563 of word embeddings using word similarity tasks. In *Proceedings of the 1st Workshop on*
11564 *Evaluating Vector-Space Representations for NLP*, Berlin, Germany, pp. 30–35. Association
11565 for Computational Linguistics.
- 11566 Fellbaum, C. (2010). *WordNet*. Springer.
- 11567 Feng, V. W., Z. Lin, and G. Hirst (2014). The impact of deep hierarchical discourse struc-
11568 tures in the evaluation of text coherence. In *Proceedings of the International Conference on*
11569 *Computational Linguistics (COLING)*, pp. 940–949.
- 11570 Feng, X., L. Huang, D. Tang, H. Ji, B. Qin, and T. Liu (2016). A language-independent
11571 neural network for event detection. In *Proceedings of the Association for Computational*
11572 *Linguistics (ACL)*, pp. 66–71.
- 11573 Fernandes, E. R., C. N. dos Santos, and R. L. Milidiú (2014). Latent trees for coreference
11574 resolution. *Computational Linguistics*.

- 11575 Ferrucci, D., E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W.
11576 Murdock, E. Nyberg, J. Prager, et al. (2010). Building Watson: An overview of the
11577 DeepQA project. *AI magazine* 31(3), 59–79.
- 11578 Ficler, J. and Y. Goldberg (2017, September). Controlling linguistic style aspects in neural
11579 language generation. In *Proceedings of the Workshop on Stylistic Variation*, Copenhagen,
11580 Denmark, pp. 94–104. Association for Computational Linguistics.
- 11581 Filippova, K. and M. Strube (2008). Sentence fusion via dependency graph compression.
11582 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 177–
11583 185.
- 11584 Fillmore, C. J. (1968). The case for case. In E. Bach and R. Harms (Eds.), *Universals in
11585 linguistic theory*. Holt, Rinehart, and Winston.
- 11586 Fillmore, C. J. (1976). Frame semantics and the nature of language. *Annals of the New York
11587 Academy of Sciences* 280(1), 20–32.
- 11588 Fillmore, C. J. and C. Baker (2009). A frames approach to semantic analysis. In *The Oxford
11589 Handbook of Linguistic Analysis*. Oxford University Press.
- 11590 Finkel, J. R., T. Grenager, and C. Manning (2005). Incorporating non-local information
11591 into information extraction systems by gibbs sampling. In *Proceedings of the Association
11592 for Computational Linguistics (ACL)*, pp. 363–370.
- 11593 Finkel, J. R., T. Grenager, and C. D. Manning (2007). The infinite tree. In *Proceedings of the
11594 Association for Computational Linguistics (ACL)*, pp. 272–279.
- 11595 Finkel, J. R., A. Kleeman, and C. D. Manning (2008). Efficient, feature-based, conditional
11596 random field parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*,
11597 pp. 959–967.
- 11598 Finkel, J. R. and C. Manning (2009). Hierarchical bayesian domain adaptation. In *Proceed-
11599 ings of the North American Chapter of the Association for Computational Linguistics (NAACL)*,
11600 pp. 602–610.
- 11601 Finkel, J. R. and C. D. Manning (2008). Enforcing transitivity in coreference resolution.
11602 In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics
11603 on Human Language Technologies: Short Papers*, pp. 45–48. Association for Computational
11604 Linguistics.
- 11605 Finkelstein, L., E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin
11606 (2002). Placing search in context: The concept revisited. *ACM Transactions on Information
11607 Systems* 20(1), 116–131.

- 11608 Firth, J. R. (1957). *Papers in Linguistics 1934-1951*. Oxford University Press.
- 11609 Flanigan, J., S. Thomson, J. Carbonell, C. Dyer, and N. A. Smith (2014). A discriminative graph-based parser for the abstract meaning representation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1426–1436.
- 11612 Foltz, P. W., W. Kintsch, and T. K. Landauer (1998). The measurement of textual coherence with latent semantic analysis. *Discourse processes* 25(2-3), 285–307.
- 11614 Fordyce, C. S. (2007). Overview of the iwslt 2007 evaluation campaign. In *International Workshop on Spoken Language Translation (IWSLT) 2007*.
- 11616 Forsyth, E. N. and C. H. Martell (2007). Lexical and discourse analysis of online chat dialog. In *International Conference on Semantic Computing*, pp. 19–26. IEEE.
- 11618 Fox, H. (2002). Phrasal cohesion and statistical machine translation. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 304–3111.
- 11620 Francis, W. and H. Kucera (1982). *Frequency analysis of English usage*. Houghton Mifflin Company.
- 11622 Francis, W. N. (1964). A standard sample of present-day English for use with digital computers. Report to the U.S Office of Education on Cooperative Research Project No. E-007.
- 11625 Freund, Y. and R. E. Schapire (1999). Large margin classification using the perceptron algorithm. *Machine learning* 37(3), 277–296.
- 11627 Fromkin, V., R. Rodman, and N. Hyams (2013). *An introduction to language*. Cengage Learning.
- 11629 Fundel, K., R. Küffner, and R. Zimmer (2007). Relex – relation extraction using dependency parse trees. *Bioinformatics* 23(3), 365–371.
- 11631 Gabow, H. N., Z. Galil, T. Spencer, and R. E. Tarjan (1986). Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica* 6(2), 109–122.
- 11634 Gabrilovich, E. and S. Markovitch (2007). Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Volume 7, pp. 1606–1611.
- 11637 Gage, P. (1994). A new algorithm for data compression. *The C Users Journal* 12(2), 23–38.

- 11638 Gale, W. A., K. W. Church, and D. Yarowsky (1992). One sense per discourse. In *Proceedings of the workshop on Speech and Natural Language*, pp. 233–237. Association for
11639 Computational Linguistics.
- 11640
- 11641 Galley, M., M. Hopkins, K. Knight, and D. Marcu (2004). What's in a translation rule? In *Proceedings of the North American Chapter of the Association for Computational Linguistics*
11642 (NAACL), pp. 273–280.
- 11643
- 11644 Galley, M., K. R. McKeown, E. Fosler-Lussier, and H. Jing (2003). Discourse segmentation
11645 of multi-party conversation. In *Proceedings of the Association for Computational Linguistics*
11646 (ACL).
- 11647 Ganchev, K. and M. Dredze (2008). Small statistical models by random feature mixing. In *Proceedings of the ACL08 HLT Workshop on Mobile Language Processing*, pp. 19–20.
- 11648
- 11649 Ganchev, K., J. Graça, J. Gillenwater, and B. Taskar (2010). Posterior regularization for
11650 structured latent variable models. *The Journal of Machine Learning Research* 11, 2001–
11651 2049.
- 11652 Ganin, Y., E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand,
11653 and V. Lempitsky (2016). Domain-adversarial training of neural networks. *Journal of*
11654 *Machine Learning Research* 17(59), 1–35.
- 11655 Gao, J., G. Andrew, M. Johnson, and K. Toutanova (2007). A comparative study of parameter
11656 estimation methods for statistical natural language processing. In *Proceedings of the*
11657 *Association for Computational Linguistics (ACL)*, pp. 824–831.
- 11658 Gatt, A. and E. Krahmer (2018). Survey of the state of the art in natural language genera-
11659 tion: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research* 61,
11660 65–170.
- 11661 Ge, D., X. Jiang, and Y. Ye (2011). A note on the complexity of l_1 p minimization. *Mathem-
11662 atical programming* 129(2), 285–299.
- 11663 Ge, N., J. Hale, and E. Charniak (1998). A statistical approach to anaphora resolution. In
11664 *Proceedings of the sixth workshop on very large corpora*, Volume 71, pp. 76.
- 11665 Ge, R., F. Huang, C. Jin, and Y. Yuan (2015). Escaping from saddle points — online stochas-
11666 tic gradient for tensor decomposition. In P. Grünwald, E. Hazan, and S. Kale (Eds.),
11667 *Proceedings of the Conference On Learning Theory (COLT)*.
- 11668 Ge, R. and R. J. Mooney (2005). A statistical semantic parser that integrates syntax and
11669 semantics. In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp.
11670 9–16.

- 11671 Geach, P. T. (1962). *Reference and generality: An examination of some medieval and modern*
11672 *theories*. Cornell University Press.
- 11673 Gehring, J., M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin (2017, May). Convolutional
11674 sequence to sequence learning. *ArXiv e-prints*.
- 11675 Gildea, D. and D. Jurafsky (2002). Automatic labeling of semantic roles. *Computational*
11676 *linguistics* 28(3), 245–288.
- 11677 Gimpel, K., N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yo-
11678 gatama, J. Flanigan, and N. A. Smith (2011). Part-of-speech tagging for Twitter: an-
11679 notation, features, and experiments. In *Proceedings of the Association for Computational*
11680 *Linguistics (ACL)*, pp. 42–47.
- 11681 Glass, J., T. J. Hazen, S. Cyphers, I. Malioutov, D. Huynh, and R. Barzilay (2007). Recent
11682 progress in the mit spoken lecture processing project. In *Eighth Annual Conference of the*
11683 *International Speech Communication Association*.
- 11684 Glorot, X. and Y. Bengio (2010). Understanding the difficulty of training deep feedforward
11685 neural networks. In *Proceedings of Artificial Intelligence and Statistics (AISTATS)*, pp. 249–
11686 256.
- 11687 Glorot, X., A. Bordes, and Y. Bengio (2011). Deep sparse rectifier networks. In *Proceedings*
11688 *of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP*
11689 *Volume*, Volume 15, pp. 315–323.
- 11690 Godfrey, J. J., E. C. Holliman, and J. McDaniel (1992). Switchboard: Telephone speech
11691 corpus for research and development. In *Proceedings of the International Conference on*
11692 *Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 517–520. IEEE.
- 11693 Goldberg, Y. (2017a, June). An adversarial review of “adversarial generation of
11694 natural language”. <https://medium.com/@yoav.goldberg/an-adversarial-review-of->
11695 [adversarial-generation-of-natural-language-409ac3378bd7](https://medium.com/@yoav.goldberg/an-adversarial-review-of-natural-language-409ac3378bd7).
- 11696 Goldberg, Y. (2017b). *Neural Network Methods for Natural Language Processing*. Synthesis
11697 Lectures on Human Language Technologies. Morgan & Claypool Publishers.
- 11698 Goldberg, Y. and M. Elhadad (2010). An efficient algorithm for easy-first non-directional
11699 dependency parsing. In *Proceedings of the North American Chapter of the Association for*
11700 *Computational Linguistics (NAACL)*, pp. 742–750.
- 11701 Goldberg, Y. and J. Nivre (2012). A dynamic oracle for arc-eager dependency parsing.
11702 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.
11703 959–976.

- 11704 Goldberg, Y., K. Zhao, and L. Huang (2013). Efficient implementation of beam-search
11705 incremental parsers. In *ACL* (2), pp. 628–633.
- 11706 Goldwater, S. and T. Griffiths (2007). A fully bayesian approach to unsupervised part-of-
11707 speech tagging. In *Annual meeting-association for computational linguistics*, Volume 45.
- 11708 Gonçalo Oliveira, H. R., F. A. Cardoso, and F. C. Pereira (2007). Tra-la-lyrics: An approach
11709 to generate text based on rhythm. In *Proceedings of the 4th. International Joint Workshop*
11710 on *Computational Creativity*. A. Cardoso and G. Wiggins.
- 11711 Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep learning*. MIT Press.
- 11712 Goodman, J. T. (2001). A bit of progress in language modeling. *Computer Speech & Lan-*
11713 *guage* 15(4), 403–434.
- 11714 Gouws, S., D. Metzler, C. Cai, and E. Hovy (2011). Contextual bearing on linguistic varia-
11715 tion in social media. In *LASM*.
- 11716 Goyal, A., H. Daume III, and S. Venkatasubramanian (2009). Streaming for large scale
11717 nlp: Language modeling. In *Proceedings of the North American Chapter of the Association*
11718 for *Computational Linguistics* (NAACL), pp. 512–520.
- 11719 Graves, A. (2012). Sequence transduction with recurrent neural networks. In *Proceedings*
11720 of the *International Conference on Machine Learning* (ICML).
- 11721 Graves, A. and N. Jaitly (2014). Towards end-to-end speech recognition with recur-
11722 rent neural networks. In *Proceedings of the International Conference on Machine Learning*
11723 (ICML), pp. 1764–1772.
- 11724 Graves, A. and J. Schmidhuber (2005). Framewise phoneme classification with bidirec-
11725 tional lstm and other neural network architectures. *Neural Networks* 18(5), 602–610.
- 11726 Grice, H. P. (1975). Logic and conversation. In P. Cole and J. L. Morgan (Eds.), *Syntax and*
11727 *Semantics Volume 3: Speech Acts*, pp. 41–58. Academic Press.
- 11728 Grishman, R. (2012). Information extraction: Capabilities and challenges. Notes prepared
11729 for the 2012 International Winter School in Language and Speech Technologies, Rovira
11730 i Virgili University, Tarragona, Spain.
- 11731 Grishman, R. (2015). Information extraction. *IEEE Intelligent Systems* 30(5), 8–15.
- 11732 Grishman, R., C. Macleod, and J. Sterling (1992). Evaluating parsing strategies using
11733 standardized parse files. In *Proceedings of the third conference on Applied natural language*
11734 *processing*, pp. 156–161. Association for Computational Linguistics.

- 11735 Grishman, R. and B. Sundheim (1996). Message understanding conference-6: A brief history. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp. 466–471.
- 11738 Groenendijk, J. and M. Stokhof (1991). Dynamic predicate logic. *Linguistics and philosophy* 14(1), 39–100.
- 11740 Grosz, B. J. (1979). Focusing and description in natural language dialogues. Technical report, SRI INTERNATIONAL MENLO PARK CA.
- 11742 Grosz, B. J., S. Weinstein, and A. K. Joshi (1995). Centering: A framework for modeling the local coherence of discourse. *Computational linguistics* 21(2), 203–225.
- 11744 Gu, J., Z. Lu, H. Li, and V. O. Li (2016). Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1631–1640.
- 11747 Gulcehre, C., S. Ahn, R. Nallapati, B. Zhou, and Y. Bengio (2016). Pointing the unknown words. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 140–149.
- 11749 Gutmann, M. U. and A. Hyvärinen (2012). Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *The Journal of Machine Learning Research* 13(1), 307–361.
- 11752 Haghghi, A. and D. Klein (2007). Unsupervised coreference resolution in a nonparametric bayesian model. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11754 Haghghi, A. and D. Klein (2009). Simple coreference resolution with rich syntactic and semantic features. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1152–1161.
- 11757 Haghghi, A. and D. Klein (2010). Coreference resolution in a modular, entity-centered model. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 385–393.
- 11760 Hajič, J. and B. Hladká (1998). Tagging inflective languages: Prediction of morphological categories for a rich, structured tagset. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 483–490.
- 11763 Halliday, M. and R. Hasan (1976). *Cohesion in English*. London: Longman.
- 11764 Hammerton, J. (2003). Named entity recognition with long short-term memory. In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 172–175.
- 11766 Han, X. and L. Sun (2012). An entity-topic model for entity linking. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 105–115.

- 11768 Han, X., L. Sun, and J. Zhao (2011). Collective entity linking in web text: a graph-based
11769 method. In *Proceedings of ACM SIGIR conference on Research and development in informa-*
11770 *tion retrieval*, pp. 765–774.
- 11771 Hannak, A., E. Anderson, L. F. Barrett, S. Lehmann, A. Mislove, and M. Riedewald (2012).
11772 Tweetin’ in the rain: Exploring societal-scale effects of weather on mood. In *Proceedings*
11773 *of the International Conference on Web and Social Media (ICWSM)*.
- 11774 Hardmeier, C. (2012). Discourse in statistical machine translation. a survey and a case
11775 study. *Discours. Revue de linguistique, psycholinguistique et informatique. A journal of lin-*
11776 *guistics, psycholinguistics and computational linguistics* (11).
- 11777 Haspelmath, M. and A. Sims (2013). *Understanding morphology*. Routledge.
- 11778 Hastie, T., R. Tibshirani, and J. Friedman (2009). *The elements of statistical learning* (Second
11779 ed.). New York: Springer.
- 11780 Hatzivassiloglou, V. and K. R. McKeown (1997). Predicting the semantic orientation of
11781 adjectives. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 174–
11782 181.
- 11783 Hayes, A. F. and K. Krippendorff (2007). Answering the call for a standard reliability
11784 measure for coding data. *Communication methods and measures* 1(1), 77–89.
- 11785 He, H., A. Balakrishnan, M. Eric, and P. Liang (2017). Learning symmetric collaborative
11786 dialogue agents with dynamic knowledge graph embeddings. In *Proceedings of the As-*
11787 *sociation for Computational Linguistics (ACL)*, pp. 1766–1776.
- 11788 He, K., X. Zhang, S. Ren, and J. Sun (2015). Delving deep into rectifiers: Surpassing
11789 human-level performance on imagenet classification. In *Proceedings of the International*
11790 *Conference on Computer Vision (ICCV)*, pp. 1026–1034.
- 11791 He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep residual learning for image recognition.
11792 In *Proceedings of the International Conference on Computer Vision (ICCV)*, pp. 770–778.
- 11793 He, L., K. Lee, M. Lewis, and L. Zettlemoyer (2017). Deep semantic role labeling: What
11794 works and what’s next. In *Proceedings of the Association for Computational Linguistics*
11795 *(ACL)*.
- 11796 He, Z., S. Liu, M. Li, M. Zhou, L. Zhang, and H. Wang (2013). Learning entity repre-
11797 sentation for entity disambiguation. In *Proceedings of the Association for Computational*
11798 *Linguistics (ACL)*, pp. 30–34.
- 11799 Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. In
11800 *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp. 539–
11801 545. Association for Computational Linguistics.

- 11802 Hearst, M. A. (1997). Texttiling: Segmenting text into multi-paragraph subtopic passages.
11803 *Computational linguistics* 23(1), 33–64.
- 11804 Heerschap, B., F. Goossen, A. Hogenboom, F. Frasincar, U. Kaymak, and F. de Jong (2011).
11805 Polarity analysis of texts using discourse structure. In *Proceedings of the 20th ACM inter-*
11806 *national conference on Information and knowledge management*, pp. 1061–1070. ACM.
- 11807 Henderson, J. (2004). Discriminative training of a neural network statistical parser. In
11808 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 95–102.
- 11809 Hendrickx, I., S. N. Kim, Z. Kozareva, P. Nakov, D. Ó Séaghdha, S. Padó, M. Pennacchiotti,
11810 L. Romano, and S. Szpakowicz (2009). Semeval-2010 task 8: Multi-way classification of
11811 semantic relations between pairs of nominals. In *Proceedings of the Workshop on Semantic*
11812 *Evaluations: Recent Achievements and Future Directions*, pp. 94–99. Association for Com-
11813 putational Linguistics.
- 11814 Hermann, K. M., T. Kočiský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and
11815 P. Blunsom (2015). Teaching machines to read and comprehend. In *Advances in Neu-*
11816 *ral Information Processing Systems*, pp. 1693–1701.
- 11817 Hernault, H., H. Prendinger, D. A. duVerle, and M. Ishizuka (2010). HILDA: A discourse
11818 parser using support vector machine classification. *Dialogue and Discourse* 1(3), 1–33.
- 11819 Hill, F., A. Bordes, S. Chopra, and J. Weston (2016). The goldilocks principle: Reading
11820 children’s books with explicit memory representations. In *Proceedings of the International*
11821 *Conference on Learning Representations (ICLR)*.
- 11822 Hill, F., K. Cho, and A. Korhonen (2016). Learning distributed representations of sentences
11823 from unlabelled data. In *Proceedings of the North American Chapter of the Association for*
11824 *Computational Linguistics (NAACL)*.
- 11825 Hindle, D. and M. Rooth (1993). Structural ambiguity and lexical relations. *Computational*
11826 *linguistics* 19(1), 103–120.
- 11827 Hirao, T., Y. Yoshida, M. Nishino, N. Yasuda, and M. Nagata (2013). Single-document
11828 summarization as a tree knapsack problem. In *Proceedings of Empirical Methods for Nat-*
11829 *ural Language Processing (EMNLP)*, pp. 1515–1520.
- 11830 Hirschman, L. and R. Gaizauskas (2001). Natural language question answering: the view
11831 from here. *natural language engineering* 7(4), 275–300.
- 11832 Hirschman, L., M. Light, E. Breck, and J. D. Burger (1999). Deep read: A reading compre-
11833 hension system. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
11834 325–332.

- 11835 Hobbs, J. R. (1978). Resolving pronoun references. *Lingua* 44(4), 311–338.
- 11836 Hobbs, J. R., D. Appelt, J. Bear, D. Israel, M. Kameyama, M. Stickel, and M. Tyson (1997).
- 11837 Fastus: A cascaded finite-state transducer for extracting information from natural-
- 11838 language text. *Finite-state language processing*, 383–406.
- 11839 Hochreiter, S. and J. Schmidhuber (1997). Long short-term memory. *Neural computa-*
- 11840 *tion* 9(8), 1735–1780.
- 11841 Hockenmaier, J. and M. Steedman (2007). Ccgbank: a corpus of ccg derivations and de-
- 11842 pendency structures extracted from the penn treebank. *Computational Linguistics* 33(3),
- 11843 355–396.
- 11844 Hoffart, J., M. A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, B. Taneva,
- 11845 S. Thater, and G. Weikum (2011). Robust disambiguation of named entities in text. In
- 11846 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 782–792.
- 11847 Hoffmann, R., C. Zhang, X. Ling, L. Zettlemoyer, and D. S. Weld (2011). Knowledge-based
- 11848 weak supervision for information extraction of overlapping relations. In *Proceedings of*
- 11849 *the Association for Computational Linguistics (ACL)*, pp. 541–550.
- 11850 Holmstrom, L. and P. Koistinen (1992). Using additive noise in back-propagation training.
- 11851 *IEEE Transactions on Neural Networks* 3(1), 24–38.
- 11852 Hovy, E. and J. Lavid (2010). Towards a ‘science’ of corpus annotation: a new methodo-
- 11853 logical challenge for corpus linguistics. *International journal of translation* 22(1), 13–36.
- 11854 Hsu, D., S. M. Kakade, and T. Zhang (2012). A spectral algorithm for learning hidden
- 11855 markov models. *Journal of Computer and System Sciences* 78(5), 1460–1480.
- 11856 Hu, M. and B. Liu (2004). Mining and summarizing customer reviews. In *Proceedings of*
- 11857 *Knowledge Discovery and Data Mining (KDD)*, pp. 168–177.
- 11858 Hu, Z., Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing (2017). Toward controlled
- 11859 generation of text. In *International Conference on Machine Learning*, pp. 1587–1596.
- 11860 Huang, F. and A. Yates (2012). Biased representation learning for domain adaptation. In
- 11861 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1313–1323.
- 11862 Huang, L. and D. Chiang (2007). Forest rescoring: Faster decoding with integrated lan-
- 11863 guage models. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
- 11864 144–151.
- 11865 Huang, L., S. Fayong, and Y. Guo (2012). Structured perceptron with inexact search. In
- 11866 *Proceedings of the North American Chapter of the Association for Computational Linguistics*
- 11867 (*NAACL*), pp. 142–151.

- 11868 Huang, Y. (2015). *Pragmatics* (Second ed.). Oxford Textbooks in Linguistics. Oxford University Press.
- 11870 Huang, Z., W. Xu, and K. Yu (2015). Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.
- 11872 Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE* 40(9), 1098–1101.
- 11874 Humphreys, K., R. Gaizauskas, and S. Azzam (1997). Event coreference for information extraction. In *Proceedings of a Workshop on Operational Factors in Practical, Robust Anaphora Resolution for Unrestricted Texts*, pp. 75–81. Association for Computational Linguistics.
- 11877 Ide, N. and Y. Wilks (2006). Making sense about sense. In *Word sense disambiguation*, pp. 47–73. Springer.
- 11879 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.
- 11882 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.
- 11885 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.
- 11888 James, G., D. Witten, T. Hastie, and R. Tibshirani (2013). *An introduction to statistical learning*, Volume 112. Springer.
- 11890 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.
- 11894 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.
- 11897 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.

- 11900 Ji, H. and R. Grishman (2011). Knowledge base population: Successful approaches and
11901 challenges. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1148–
11902 1158.
- 11903 Ji, Y., T. Cohn, L. Kong, C. Dyer, and J. Eisenstein (2015). Document context language
11904 models. In *International Conference on Learning Representations, Workshop Track*, Volume
11905 abs/1511.03962.
- 11906 Ji, Y. and J. Eisenstein (2014). Representation learning for text-level discourse parsing. In
11907 *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11908 Ji, Y. and J. Eisenstein (2015, June). One vector is not enough: Entity-augmented distribu-
11909 tional semantics for discourse relations. *Transactions of the Association for Computational
11910 Linguistics (TACL)*.
- 11911 Ji, Y., G. Haffari, and J. Eisenstein (2016). A latent variable recurrent neural network for
11912 discourse relation language models. In *Proceedings of the North American Chapter of the
11913 Association for Computational Linguistics (NAACL)*.
- 11914 Ji, Y. and N. A. Smith (2017). Neural discourse structure for text categorization. In *Pro-
11915 ceedings of the Association for Computational Linguistics (ACL)*, pp. 996–1005.
- 11916 Ji, Y., C. Tan, S. Martschat, Y. Choi, and N. A. Smith (2017). Dynamic entity representations
11917 in neural language models. In *Proceedings of Empirical Methods for Natural Language
11918 Processing (EMNLP)*, pp. 1831–1840.
- 11919 Jiang, L., M. Yu, M. Zhou, X. Liu, and T. Zhao (2011). Target-dependent twitter sentiment
11920 classification. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
11921 151–160.
- 11922 Jing, H. (2000). Sentence reduction for automatic text summarization. In *Proceedings of
11923 the sixth conference on Applied natural language processing*, pp. 310–315. Association for
11924 Computational Linguistics.
- 11925 Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of
11926 Knowledge Discovery and Data Mining (KDD)*, pp. 133–142.
- 11927 Jockers, M. L. (2015). Szuzhet? <http://bla.bla.com>.
- 11928 Johnson, A. E., T. J. Pollard, L. Shen, H. L. Li-wei, M. Feng, M. Ghassemi, B. Moody,
11929 P. Szolovits, L. A. Celi, and R. G. Mark (2016). Mimic-iii, a freely accessible critical care
11930 database. *Scientific data* 3, 160035.
- 11931 Johnson, M. (1998). Pcfg models of linguistic tree representations. *Computational Linguis-
11932 tics* 24(4), 613–632.

- 11933 Johnson, R. and T. Zhang (2017). Deep pyramid convolutional neural networks for text
 11934 categorization. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
 11935 562–570.
- 11936 Joshi, A. K. (1985). How much context-sensitivity is required to provide reasonable struc-
 11937 tural descriptions? – tree adjoining grammars. In *Natural Language Processing – Theoret-
 11938 ical, Computational and Psychological Perspective*. New York, NY: Cambridge University
 11939 Press.
- 11940 Joshi, A. K. and Y. Schabes (1997). Tree-adjoining grammars. In *Handbook of formal lan-
 11941 guages*, pp. 69–123. Springer.
- 11942 Joshi, A. K., K. V. Shanker, and D. Weir (1991). The convergence of mildly context-sensitive
 11943 grammar formalisms. In *Foundational Issues in Natural Language Processing*. Cambridge
 11944 MA: MIT Press.
- 11945 Jozefowicz, R., O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu (2016). Exploring the limits
 11946 of language modeling. *arXiv preprint arXiv:1602.02410*.
- 11947 Jozefowicz, R., W. Zaremba, and I. Sutskever (2015). An empirical exploration of recurrent
 11948 network architectures. In *Proceedings of the International Conference on Machine Learning
 11949 (ICML)*, pp. 2342–2350.
- 11950 Jurafsky, D. (1996). A probabilistic model of lexical and syntactic access and disambigua-
 11951 tion. *Cognitive Science* 20(2), 137–194.
- 11952 Jurafsky, D. and J. H. Martin (2009). *Speech and Language Processing* (Second ed.). Prentice
 11953 Hall.
- 11954 Jurafsky, D. and J. H. Martin (2018). *Speech and Language Processing* (Third ed.). Prentice
 11955 Hall.
- 11956 Kadlec, R., M. Schmid, O. Bajgar, and J. Kleindienst (2016). Text understanding with
 11957 the attention sum reader network. In *Proceedings of the Association for Computational
 11958 Linguistics (ACL)*, pp. 908–918.
- 11959 Kalchbrenner, N. and P. Blunsom (2013, August). Recurrent convolutional neural net-
 11960 works for discourse compositionality. In *Proceedings of the Workshop on Continuous Vec-
 11961 tor Space Models and their Compositionality*, Sofia, Bulgaria, pp. 119–126. Association for
 11962 Computational Linguistics.
- 11963 Kalchbrenner, N., L. Espeholt, K. Simonyan, A. v. d. Oord, A. Graves, and K. Kavukcuoglu
 11964 (2016). Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*.

- 11965 Kalchbrenner, N., E. Grefenstette, and P. Blunsom (2014). A convolutional neural network
11966 for modelling sentences. In *Proceedings of the Association for Computational Linguistics*
11967 (*ACL*), pp. 655–665.
- 11968 Karlsson, F. (2007). Constraints on multiple center-embedding of clauses. *Journal of Lin-*
11969 *guistics* 43(02), 365–392.
- 11970 Kate, R. J., Y. W. Wong, and R. J. Mooney (2005). Learning to transform natural to formal
11971 languages. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.
- 11972 Kehler, A. (2007). Rethinking the SMASH approach to pronoun interpretation. In *Interdis-*
11973 *ciplinary perspectives on reference processing*, New Directions in Cognitive Science Series,
11974 pp. 95–122. Oxford University Press.
- 11975 Kibble, R. and R. Power (2004). Optimizing referential coherence in text generation. *Com-*
11976 *putational Linguistics* 30(4), 401–416.
- 11977 Kilgarriff, A. (1997). I don't believe in word senses. *Computers and the Humanities* 31(2),
11978 91–113.
- 11979 Kilgarriff, A. and G. Grefenstette (2003). Introduction to the special issue on the web as
11980 corpus. *Computational linguistics* 29(3), 333–347.
- 11981 Kim, M.-J. (2002). Does korean have adjectives? *MIT Working Papers in Linguistics* 43,
11982 71–89.
- 11983 Kim, S.-M. and E. Hovy (2006, July). Extracting opinions, opinion holders, and topics
11984 expressed in online news media text. In *Proceedings of the Workshop on Sentiment and*
11985 *Subjectivity in Text*, Sydney, Australia, pp. 1–8. Association for Computational Linguis-
11986 tics.
- 11987 Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings*
11988 *of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1746–1751.
- 11989 Kim, Y., C. Denton, L. Hoang, and A. M. Rush (2017). Structured attention networks. In
11990 *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 11991 Kim, Y., Y. Jernite, D. Sontag, and A. M. Rush (2016). Character-aware neural language
11992 models. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.
- 11993 Kingma, D. and J. Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint*
11994 *arXiv:1412.6980*.
- 11995 Kiperwasser, E. and Y. Goldberg (2016). Simple and accurate dependency parsing using
11996 bidirectional lstm feature representations. *Transactions of the Association for Compu-*
11997 *tational Linguistics* 4, 313–327.

- 11998 Kipper-Schuler, K. (2005). *VerbNet: A broad-coverage, comprehensive verb lexicon*. Ph. D.
11999 thesis, Computer and Information Science, University of Pennsylvania.
- 12000 Kiros, R., R. Salakhutdinov, and R. Zemel (2014). Multimodal neural language models. In
12001 *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 595–603.
- 12002 Kiros, R., Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler
12003 (2015). Skip-thought vectors. In *Neural Information Processing Systems (NIPS)*.
- 12004 Klein, D. and C. D. Manning (2003). Accurate unlexicalized parsing. In *Proceedings of the*
12005 *Association for Computational Linguistics (ACL)*, pp. 423–430.
- 12006 Klein, D. and C. D. Manning (2004). Corpus-based induction of syntactic structure: Mod-
12007 els of dependency and constituency. In *Proceedings of the Association for Computational*
12008 *Linguistics (ACL)*.
- 12009 Klein, G., Y. Kim, Y. Deng, J. Senellart, and A. M. Rush (2017). Opennmt: Open-source
12010 toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*.
- 12011 Klementiev, A., I. Titov, and B. Bhattacharai (2012). Inducing crosslingual distributed repre-
12012 sentations of words. In *Proceedings of the International Conference on Computational Lin-*
12013 *guistics (COLING)*, pp. 1459–1474.
- 12014 Klenner, M. (2007). Enforcing consistency on coreference sets. In *Recent Advances in Natu-*
12015 *ral Language Processing (RANLP)*, pp. 323–328.
- 12016 Knight, K. (1999). Decoding complexity in word-replacement translation models. *Compu-*
12017 *tational Linguistics* 25(4), 607–615.
- 12018 Knight, K. and J. Graehl (1998). Machine transliteration. *Computational Linguistics* 24(4),
12019 599–612.
- 12020 Knight, K. and D. Marcu (2000). Statistics-based summarization-step one: Sentence com-
12021 pression. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp.
12022 703–710.
- 12023 Knight, K. and J. May (2009). Applications of weighted automata in natural language
12024 processing. In *Handbook of Weighted Automata*, pp. 571–596. Springer.
- 12025 Knott, A. (1996). *A data-driven methodology for motivating a set of coherence relations*. Ph. D.
12026 thesis, The University of Edinburgh.
- 12027 Koehn, P. (2005). Europarl: A parallel corpus for statistical machine translation. In *MT*
12028 *summit*, Volume 5, pp. 79–86.
- 12029 Koehn, P. (2009). *Statistical machine translation*. Cambridge University Press.

- 12030 Koehn, P. (2017). Neural machine translation. *arXiv preprint arXiv:1709.07809*.
- 12031 Konstas, I. and M. Lapata (2013). A global model for concept-to-text generation. *Journal
of Artificial Intelligence Research* 48, 305–346.
- 12033 Koo, T., X. Carreras, and M. Collins (2008, jun). Simple semi-supervised dependency
12034 parsing. In *Proceedings of ACL-08: HLT*, Columbus, Ohio, pp. 595–603. Association for
12035 Computational Linguistics.
- 12036 Koo, T. and M. Collins (2005). Hidden-variable models for discriminative reranking. In
12037 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 507–514.
- 12038 Koo, T. and M. Collins (2010). Efficient third-order dependency parsers. In *Proceedings of
the Association for Computational Linguistics (ACL)*.
- 12040 Koo, T., A. Globerson, X. Carreras, and M. Collins (2007). Structured prediction models
12041 via the matrix-tree theorem. In *Proceedings of Empirical Methods for Natural Language
12042 Processing (EMNLP)*, pp. 141–150.
- 12043 Kovach, B. and T. Rosenstiel (2014). *The elements of journalism: What newpeople should know
12044 and the public should expect*. Three Rivers Press.
- 12045 Krishnamurthy, J. (2016). Probabilistic models for learning a semantic parser lexicon. In
12046 *Proceedings of the North American Chapter of the Association for Computational Linguistics
12047 (NAACL)*, pp. 606–616.
- 12048 Krishnamurthy, J. and T. M. Mitchell (2012). Weakly supervised training of semantic
12049 parsers. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
12050 pp. 754–765.
- 12051 Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep
12052 convolutional neural networks. In *Neural Information Processing Systems (NIPS)*, pp.
12053 1097–1105.
- 12054 Kübler, S., R. McDonald, and J. Nivre (2009). Dependency parsing. *Synthesis Lectures on
12055 Human Language Technologies* 1(1), 1–127.
- 12056 Kuhlmann, M. and J. Nivre (2010). Transition-based techniques for non-projective depen-
12057 dency parsing. *Northern European Journal of Language Technology (NEJLT)* 2(1), 1–19.
- 12058 Kummerfeld, J. K., T. Berg-Kirkpatrick, and D. Klein (2015). An empirical analysis of op-
12059 timization for max-margin NLP. In *Proceedings of Empirical Methods for Natural Language
12060 Processing (EMNLP)*.

- 12061 Kwiatkowski, T., S. Goldwater, L. Zettlemoyer, and M. Steedman (2012). A probabilistic
12062 model of syntactic and semantic acquisition from child-directed utterances and their
12063 meanings. In *Proceedings of the European Chapter of the Association for Computational Lin-*
12064 *guistics (EACL)*, pp. 234–244.
- 12065 Lafferty, J., A. McCallum, and F. Pereira (2001). Conditional random fields: Probabilistic
12066 models for segmenting and labeling sequence data. In *icml*.
- 12067 Lakoff, G. (1973). Hedges: A study in meaning criteria and the logic of fuzzy concepts.
12068 *Journal of philosophical logic* 2(4), 458–508.
- 12069 Lample, G., M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer (2016). Neural
12070 architectures for named entity recognition. In *Proceedings of the North American Chapter*
12071 *of the Association for Computational Linguistics (NAACL)*, pp. 260–270.
- 12072 Langkilde, I. and K. Knight (1998). Generation that exploits corpus-based statistical
12073 knowledge. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 704–
12074 710.
- 12075 Lapata, M. (2003). Probabilistic text structuring: Experiments with sentence ordering. In
12076 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 545–552.
- 12077 Lappin, S. and H. J. Leass (1994). An algorithm for pronominal anaphora resolution.
12078 *Computational linguistics* 20(4), 535–561.
- 12079 Lari, K. and S. J. Young (1990). The estimation of stochastic context-free grammars using
12080 the inside-outside algorithm. *Computer speech & language* 4(1), 35–56.
- 12081 Lascarides, A. and N. Asher (2007). Segmented discourse representation theory: Dynamic
12082 semantics with discourse structure. In *Computing meaning*, pp. 87–124. Springer.
- 12083 Law, E. and L. v. Ahn (2011). Human computation. *Synthesis Lectures on Artificial Intelli-*
12084 *gence and Machine Learning* 5(3), 1–121.
- 12085 Lebret, R., D. Grangier, and M. Auli (2016). Neural text generation from structured data
12086 with application to the biography domain. In *Proceedings of Empirical Methods for Natural*
12087 *Language Processing (EMNLP)*, pp. 1203–1213.
- 12088 LeCun, Y., L. Bottou, G. B. Orr, and K.-R. Müller (1998). Efficient backprop. In *Neural*
12089 *networks: Tricks of the trade*, pp. 9–50. Springer.
- 12090 Lee, C. M. and S. S. Narayanan (2005). Toward detecting emotions in spoken dialogs.
12091 *IEEE transactions on speech and audio processing* 13(2), 293–303.

- 12092 Lee, H., A. Chang, Y. Peirsman, N. Chambers, M. Surdeanu, and D. Jurafsky (2013). De-
12093 terministic coreference resolution based on entity-centric, precision-ranked rules. *Com-*
12094 *putational Linguistics* 39(4), 885–916.
- 12095 Lee, H., Y. Peirsman, A. Chang, N. Chambers, M. Surdeanu, and D. Jurafsky (2011). Stan-
12096 ford’s multi-pass sieve coreference resolution system at the conll-2011 shared task. In
12097 *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 28–34. Associa-
12098 tion for Computational Linguistics.
- 12099 Lee, K., L. He, M. Lewis, and L. Zettlemoyer (2017). End-to-end neural coreference reso-
12100 lution. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12101 Lenat, D. B., R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd (1990). Cyc: toward
12102 programs with common sense. *Communications of the ACM* 33(8), 30–49.
- 12103 Lesk, M. (1986). Automatic sense disambiguation using machine readable dictionaries:
12104 how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual interna-*
12105 *tional conference on Systems documentation*, pp. 24–26. ACM.
- 12106 Levesque, H. J., E. Davis, and L. Morgenstern (2011). The winograd schema challenge.
12107 In *Aaa spring symposium: Logical formalizations of commonsense reasoning*, Volume 46, pp.
12108 47.
- 12109 Levin, E., R. Pieraccini, and W. Eckert (1998). Using markov decision process for learning
12110 dialogue strategies. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the*
12111 *1998 IEEE International Conference on*, Volume 1, pp. 201–204. IEEE.
- 12112 Levy, O. and Y. Goldberg (2014). Dependency-based word embeddings. In *Proceedings of*
12113 *the Association for Computational Linguistics (ACL)*, pp. 302–308.
- 12114 Levy, O., Y. Goldberg, and I. Dagan (2015). Improving distributional similarity with
12115 lessons learned from word embeddings. *Transactions of the Association for Computational*
12116 *Linguistics* 3, 211–225.
- 12117 Levy, R. and C. Manning (2009). An informal introduction to computational semantics.
- 12118 Lewis, M. and M. Steedman (2013). Combined distributional and logical semantics. *Trans-*
12119 *actions of the Association for Computational Linguistics* 1, 179–192.
- 12120 Lewis II, P. M. and R. E. Stearns (1968). Syntax-directed transduction. *Journal of the ACM*
12121 (*JACM*) 15(3), 465–488.
- 12122 Li, J. and D. Jurafsky (2015). Do multi-sense embeddings improve natural language
12123 understanding? In *Proceedings of Empirical Methods for Natural Language Processing*
12124 (*EMNLP*), pp. 1722–1732.

- 12125 Li, J. and D. Jurafsky (2017). Neural net models of open-domain discourse coherence. In
 12126 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 198–209.
- 12127 Li, J., R. Li, and E. Hovy (2014). Recursive deep models for discourse parsing. In *Proceed-
 12128 ings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12129 Li, J., M.-T. Luong, and D. Jurafsky (2015). A hierarchical neural autoencoder for para-
 12130 graphs and documents. In *Proceedings of Empirical Methods for Natural Language Process-
 12131 ing (EMNLP)*.
- 12132 Li, J., T. Luong, D. Jurafsky, and E. Hovy (2015). When are tree structures necessary
 12133 for deep learning of representations? In *Proceedings of Empirical Methods for Natural
 12134 Language Processing (EMNLP)*, pp. 2304–2314.
- 12135 Li, J., W. Monroe, A. Ritter, D. Jurafsky, M. Galley, and J. Gao (2016, November). Deep
 12136 reinforcement learning for dialogue generation. In *Proceedings of the 2016 Conference on
 12137 Empirical Methods in Natural Language Processing*, Austin, Texas, pp. 1192–1202. Associa-
 12138 tion for Computational Linguistics.
- 12139 Li, Q., S. Anzaroot, W.-P. Lin, X. Li, and H. Ji (2011). Joint inference for cross-document
 12140 information extraction. In *Proceedings of the International Conference on Information and
 12141 Knowledge Management (CIKM)*, pp. 2225–2228.
- 12142 Li, Q., H. Ji, and L. Huang (2013). Joint event extraction via structured prediction with global
 12143 features. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
 12144 73–82.
- 12145 Liang, P., A. Bouchard-Côté, D. Klein, and B. Taskar (2006). An end-to-end discriminative
 12146 approach to machine translation. In *Proceedings of the Association for Computational
 12147 Linguistics (ACL)*, pp. 761–768.
- 12148 Liang, P., M. Jordan, and D. Klein (2009). Learning semantic correspondences with less
 12149 supervision. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 91–
 12150 99.
- 12151 Liang, P., M. I. Jordan, and D. Klein (2013). Learning dependency-based compositional
 12152 semantics. *Computational Linguistics* 39(2), 389–446.
- 12153 Liang, P. and D. Klein (2009). Online em for unsupervised models. In *Proceedings of the
 12154 North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 611–
 12155 619.
- 12156 Liang, P., S. Petrov, M. I. Jordan, and D. Klein (2007). The infinite pcfg using hierarchical
 12157 dirichlet processes. In *Proceedings of Empirical Methods for Natural Language Processing
 12158 (EMNLP)*, pp. 688–697.

- 12159 Liang, P. and C. Potts (2015). Bringing machine learning and compositional semantics
12160 together. *Annual Review of Linguistics* 1(1), 355–376.
- 12161 Lieber, R. (2015). *Introducing morphology*. Cambridge University Press.
- 12162 Lin, D. (1998). Automatic retrieval and clustering of similar words. In *Proceedings of the*
12163 *17th international conference on Computational linguistics-Volume 2*, pp. 768–774. Association
12164 for Computational Linguistics.
- 12165 Lin, J. and C. Dyer (2010). Data-intensive text processing with mapreduce. *Synthesis*
12166 *Lectures on Human Language Technologies* 3(1), 1–177.
- 12167 Lin, Z., M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio (2017). A
12168 structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*.
- 12169 Lin, Z., M.-Y. Kan, and H. T. Ng (2009). Recognizing implicit discourse relations in the
12170 penn discourse treebank. In *Proceedings of Empirical Methods for Natural Language Pro-*
12171 *cessing (EMNLP)*, pp. 343–351.
- 12172 Lin, Z., H. T. Ng, and M.-Y. Kan (2011). Automatically evaluating text coherence using
12173 discourse relations. In *Proceedings of the Association for Computational Linguistics (ACL)*,
12174 pp. 997–1006.
- 12175 Lin, Z., H. T. Ng, and M. Y. Kan (2014, nov). A PDTB-styled end-to-end discourse parser.
12176 *Natural Language Engineering FirstView*, 1–34.
- 12177 Ling, W., C. Dyer, A. Black, and I. Trancoso (2015). Two/too simple adaptations of
12178 word2vec for syntax problems. In *Proceedings of the North American Chapter of the As-*
12179 *sociation for Computational Linguistics (NAACL)*.
- 12180 Ling, W., T. Luís, L. Marujo, R. F. Astudillo, S. Amir, C. Dyer, A. W. Black, and I. Trancoso
12181 (2015). Finding function in form: Compositional character models for open vocabulary
12182 word representation. In *Proceedings of Empirical Methods for Natural Language Processing*
12183 (*EMNLP*).
- 12184 Ling, W., G. Xiang, C. Dyer, A. Black, and I. Trancoso (2013). Microblogs as parallel cor-
12185 pora. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 12186 Ling, X., S. Singh, and D. S. Weld (2015). Design challenges for entity linking. *Transactions*
12187 *of the Association for Computational Linguistics* 3, 315–328.
- 12188 Linguistic Data Consortium (2005, July). ACE (automatic content extraction) English an-
12189 notation guidelines for relations. Technical Report Version 5.8.3, Linguistic Data Con-
12190 sortium.

- 12191 Liu, B. (2015). *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. Cambridge
12192 University Press.
- 12193 Liu, D. C. and J. Nocedal (1989). On the limited memory BFGS method for large scale
12194 optimization. *Mathematical programming* 45(1-3), 503–528.
- 12195 Liu, Y., Q. Liu, and S. Lin (2006). Tree-to-string alignment template for statistical machine
12196 translation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 609–
12197 616.
- 12198 Loper, E. and S. Bird (2002). Nltk: The natural language toolkit. In *Proceedings of the ACL-*
12199 *02 Workshop on Effective tools and methodologies for teaching natural language processing and*
12200 *computational linguistics-Volume 1*, pp. 63–70. Association for Computational Linguistics.
- 12201 Louis, A., A. Joshi, and A. Nenkova (2010). Discourse indicators for content selection in
12202 summarization. In *Proceedings of the 11th Annual Meeting of the Special Interest Group on*
12203 *Discourse and Dialogue*, pp. 147–156. Association for Computational Linguistics.
- 12204 Louis, A. and A. Nenkova (2013). What makes writing great? first experiments on article
12205 quality prediction in the science journalism domain. *Transactions of the Association for*
12206 *Computational Linguistics* 1, 341–352.
- 12207 Loveland, D. W. (2016). *Automated Theorem Proving: a logical basis*. Elsevier.
- 12208 Lowe, R., N. Pow, I. V. Serban, and J. Pineau (2015). The ubuntu dialogue corpus: A large
12209 dataset for research in unstructured multi-turn dialogue systems. In *Proceedings of the*
12210 *Special Interest Group on Discourse and Dialogue (SIGDIAL)*.
- 12211 Luo, X. (2005). On coreference resolution performance metrics. In *Proceedings of Empirical*
12212 *Methods for Natural Language Processing (EMNLP)*, pp. 25–32.
- 12213 Luo, X., A. Ittycheriah, H. Jing, N. Kambhatla, and S. Roukos (2004). A mention-
12214 synchronous coreference resolution algorithm based on the bell tree. In *Proceedings*
12215 *of the Association for Computational Linguistics (ACL)*.
- 12216 Luong, M.-T., R. Socher, and C. D. Manning (2013). Better word representations with
12217 recursive neural networks for morphology. *CoNLL-2013* 104.
- 12218 Luong, T., H. Pham, and C. D. Manning (2015). Effective approaches to attention-based
12219 neural machine translation. In *Proceedings of Empirical Methods for Natural Language*
12220 *Processing (EMNLP)*, pp. 1412–1421.
- 12221 Luong, T., I. Sutskever, Q. Le, O. Vinyals, and W. Zaremba (2015). Addressing the rare
12222 word problem in neural machine translation. In *Proceedings of the Association for Compu-*
12223 *tational Linguistics (ACL)*, pp. 11–19.

- 12224 Maas, A. L., A. Y. Hannun, and A. Y. Ng (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the International Conference on Machine Learning (ICML)*.
- 12225
- 12226
- 12227 Mairesse, F. and M. A. Walker (2011). Controlling user perceptions of linguistic style: Trainable generation of personality traits. *Computational Linguistics* 37(3), 455–488.
- 12228
- 12229 Mani, I., M. Verhagen, B. Wellner, C. M. Lee, and J. Pustejovsky (2006). Machine learning of temporal relations. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 753–760.
- 12230
- 12231
- 12232 Mann, W. C. and S. A. Thompson (1988). Rhetorical structure theory: Toward a functional theory of text organization. *Text* 8(3), 243–281.
- 12233
- 12234 Manning, C. D. (2015). Computational linguistics and deep learning. *Computational Linguistics* 41(4), 701–707.
- 12235
- 12236 Manning, C. D. (2016). Computational linguistics and deep learning. *Computational Linguistics* 41(4).
- 12237
- 12238 Manning, C. D., P. Raghavan, H. Schütze, et al. (2008). *Introduction to information retrieval*, Volume 1. Cambridge university press.
- 12239
- 12240 Manning, C. D. and H. Schütze (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts: MIT press.
- 12241
- 12242 Marcu, D. (1996). Building up rhetorical structure trees. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 1069–1074.
- 12243
- 12244 Marcu, D. (1997a). From discourse structures to text summaries. In *Proceedings of the workshop on Intelligent Scalable Text Summarization*.
- 12245
- 12246 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.
- 12247
- 12248 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.
- 12249
- 12250 Maron, O. and T. Lozano-Pérez (1998). A framework for multiple-instance learning. In *Neural Information Processing Systems (NIPS)*, pp. 570–576.
- 12251
- 12252 Márquez, G. G. (1970). *One Hundred Years of Solitude*. Harper & Row. English translation by Gregory Rabassa.
- 12253

- 12254 Martins, A. F. T., N. A. Smith, and E. P. Xing (2009). Concise integer linear programming
 12255 formulations for dependency parsing. In *Proceedings of the Association for Computational
 12256 Linguistics (ACL)*, pp. 342–350.
- 12257 Martins, A. F. T., N. A. Smith, E. P. Xing, P. M. Q. Aguiar, and M. A. T. Figueiredo (2010).
 12258 Turbo parsers: Dependency parsing by approximate variational inference. In *Proceed-
 12259 ings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 34–44.
- 12260 Matsuzaki, T., Y. Miyao, and J. Tsujii (2005). Probabilistic cfg with latent annotations. In
 12261 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 75–82.
- 12262 Matthiessen, C. and J. A. Bateman (1991). *Text generation and systemic-functional linguistics:
 12263 experiences from English and Japanese*. Pinter Publishers.
- 12264 McCallum, A. and W. Li (2003). Early results for named entity recognition with condi-
 12265 tional random fields, feature induction and web-enhanced lexicons. In *Proceedings of
 12266 the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp.
 12267 188–191.
- 12268 McCallum, A. and B. Wellner (2004). Conditional models of identity uncertainty with
 12269 application to noun coreference. In *NIPS*, pp. 905–912.
- 12270 McDonald, R., K. Crammer, and F. Pereira (2005). Online large-margin training of depen-
 12271 dency parsers. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
 12272 91–98.
- 12273 McDonald, R., K. Hannan, T. Neylon, M. Wells, and J. Reynar (2007). Structured models
 12274 for fine-to-coarse sentiment analysis. In *Proceedings of ACL*.
- 12275 McDonald, R. and F. Pereira (2006). Online learning of approximate dependency parsing
 12276 algorithms. In *Proceedings of the European Chapter of the Association for Computational
 12277 Linguistics (EACL)*.
- 12278 McKeown, K. (1992). *Text generation*. Cambridge University Press.
- 12279 McKeown, K., S. Rosenthal, K. Thadani, and C. Moore (2010). Time-efficient creation of
 12280 an accurate sentence fusion corpus. In *Proceedings of the North American Chapter of the
 12281 Association for Computational Linguistics (NAACL)*, pp. 317–320.
- 12282 McKeown, K. R., R. Barzilay, D. Evans, V. Hatzivassiloglou, J. L. Klavans, A. Nenkova,
 12283 C. Sable, B. Schiffman, and S. Sigelman (2002). Tracking and summarizing news on a
 12284 daily basis with columbia’s newsblaster. In *Proceedings of the second international confer-
 12285 ence on Human Language Technology Research*, pp. 280–285.

- 12286 McNamee, P. and H. T. Dang (2009). Overview of the tac 2009 knowledge base population
12287 track. In *Text Analysis Conference (TAC)*, Volume 17, pp. 111–113.
- 12288 Medlock, B. and T. Briscoe (2007). Weakly supervised learning for hedge classification in
12289 scientific literature. In *Proceedings of the Association for Computational Linguistics (ACL)*,
12290 pp. 992–999.
- 12291 Mei, H., M. Bansal, and M. R. Walter (2016). What to talk about and how? selective gen-
12292 eration using lstms with coarse-to-fine alignment. In *Proceedings of the North American*
12293 *Chapter of the Association for Computational Linguistics (NAACL)*, pp. 720–730.
- 12294 Merity, S., N. S. Keskar, and R. Socher (2018). Regularizing and optimizing lstm language
12295 models. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 12296 Merity, S., C. Xiong, J. Bradbury, and R. Socher (2017). Pointer sentinel mixture models.
12297 In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 12298 Messud, C. (2014, June). A new 'l'étranger'. *New York Review of Books*.
- 12299 Miao, Y. and P. Blunsom (2016). Language as a latent variable: Discrete generative mod-
12300 els for sentence compression. In *Proceedings of Empirical Methods for Natural Language*
12301 *Processing (EMNLP)*, pp. 319–328.
- 12302 Miao, Y., L. Yu, and P. Blunsom (2016). Neural variational inference for text processing. In
12303 *Proceedings of the International Conference on Machine Learning (ICML)*.
- 12304 Mihalcea, R., T. A. Chklovski, and A. Kilgarriff (2004, July). The senseval-3 english lexical
12305 sample task. In *Proceedings of SENSEVAL-3*, Barcelona, Spain, pp. 25–28. Association for
12306 Computational Linguistics.
- 12307 Mihalcea, R. and D. Radev (2011). *Graph-based natural language processing and information*
12308 *retrieval*. Cambridge University Press.
- 12309 Mikolov, T., K. Chen, G. Corrado, and J. Dean (2013). Efficient estimation of word repre-
12310 sentations in vector space. In *Proceedings of International Conference on Learning Represen-*
12311 *tations*.
- 12312 Mikolov, T., A. Deoras, D. Povey, L. Burget, and J. Cernocky (2011). Strategies for train-
12313 ing large scale neural network language models. In *Automatic Speech Recognition and*
12314 *Understanding (ASRU), 2011 IEEE Workshop on*, pp. 196–201. IEEE.
- 12315 Mikolov, T., M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur (2010). Recurrent
12316 neural network based language model. In *INTERSPEECH*, pp. 1045–1048.

- 12317 Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013). Distributed rep-
12318 resentations of words and phrases and their compositionality. In *Advances in Neural*
12319 *Information Processing Systems*, pp. 3111–3119.
- 12320 Mikolov, T., W.-t. Yih, and G. Zweig (2013). Linguistic regularities in continuous space
12321 word representations. In *Proceedings of the North American Chapter of the Association for*
12322 *Computational Linguistics (NAACL)*, pp. 746–751.
- 12323 Mikolov, T. and G. Zweig. Context dependent recurrent neural network language model.
12324 In *Proceedings of Spoken Language Technology (SLT)*, pp. 234–239.
- 12325 Miller, G. A., G. A. Heise, and W. Lichten (1951). The intelligibility of speech as a function
12326 of the context of the test materials. *Journal of experimental psychology* 41(5), 329.
- 12327 Miller, M., C. Sathi, D. Wiesenthal, J. Leskovec, and C. Potts (2011). Sentiment flow
12328 through hyperlink networks. In *Proceedings of the International Conference on Web and*
12329 *Social Media (ICWSM)*.
- 12330 Miller, S., J. Guinness, and A. Zamanian (2004). Name tagging with word clusters and
12331 discriminative training. In *Proceedings of the North American Chapter of the Association for*
12332 *Computational Linguistics (NAACL)*, pp. 337–342.
- 12333 Milne, D. and I. H. Witten (2008). Learning to link with wikipedia. In *Proceedings of the*
12334 *International Conference on Information and Knowledge Management (CIKM)*, pp. 509–518.
12335 ACM.
- 12336 Miltsakaki, E. and K. Kukich (2004). Evaluation of text coherence for electronic essay
12337 scoring systems. *Natural Language Engineering* 10(1), 25–55.
- 12338 Minka, T. P. (1999). From hidden markov models to linear dynamical systems. Tech. Rep.
12339 531, Vision and Modeling Group of Media Lab, MIT.
- 12340 Minsky, M. (1974). A framework for representing knowledge. Technical Report 306, MIT
12341 AI Laboratory.
- 12342 Minsky, M. and S. Papert (1969). *Perceptrons*. MIT press.
- 12343 Mintz, M., S. Bills, R. Snow, and D. Jurafsky (2009). Distant supervision for relation extrac-
12344 tion without labeled data. In *Proceedings of the Association for Computational Linguistics*
12345 (*ACL*), pp. 1003–1011.
- 12346 Mirza, P., R. Sprugnoli, S. Tonelli, and M. Speranza (2014). Annotating causality in the
12347 tempeval-3 corpus. In *Proceedings of the EACL 2014 Workshop on Computational Ap-*
12348 *proaches to Causality in Language (CAtoCL)*, pp. 10–19.

- 12349 Misra, D. K. and Y. Artzi (2016). Neural shift-reduce ccg semantic parsing. In *Proceedings*
12350 *of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12351 Mitchell, J. and M. Lapata (2010). Composition in distributional models of semantics.
12352 *Cognitive Science* 34(8), 1388–1429.
- 12353 Miwa, M. and M. Bansal (2016). End-to-end relation extraction using lstms on sequences
12354 and tree structures. In *Proceedings of the Association for Computational Linguistics (ACL)*,
12355 pp. 1105–1116.
- 12356 Mnih, A. and G. Hinton (2007). Three new graphical models for statistical language mod-
12357 ellng. In *Proceedings of the 24th international conference on Machine learning, ICML '07*,
12358 New York, NY, USA, pp. 641–648. ACM.
- 12359 Mnih, A. and G. E. Hinton (2008). A scalable hierarchical distributed language model. In
12360 *Neural Information Processing Systems (NIPS)*, pp. 1081–1088.
- 12361 Mnih, A. and Y. W. Teh (2012). A fast and simple algorithm for training neural probabilis-
12362 tic language models. In *Proceedings of the International Conference on Machine Learning*
12363 (*ICML*).
- 12364 Mohammad, S. M. and P. D. Turney (2013). Crowdsourcing a word–emotion association
12365 lexicon. *Computational Intelligence* 29(3), 436–465.
- 12366 Mohri, M., F. Pereira, and M. Riley (2002). Weighted finite-state transducers in speech
12367 recognition. *Computer Speech & Language* 16(1), 69–88.
- 12368 Mohri, M., A. Rostamizadeh, and A. Talwalkar (2012). *Foundations of machine learning*.
12369 MIT press.
- 12370 Montague, R. (1973). The proper treatment of quantification in ordinary english. In *Ap-
12371 proaches to natural language*, pp. 221–242. Springer.
- 12372 Moore, J. D. and C. L. Paris (1993, dec). Planning text for advisory dialogues: Capturing
12373 intentional and rhetorical information. *Comput. Linguist.* 19(4), 651–694.
- 12374 Morante, R. and E. Blanco (2012). *sem 2012 shared task: Resolving the scope and fo-
12375 cus of negation. In *Proceedings of the First Joint Conference on Lexical and Computational*
12376 *Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2:*
12377 *Proceedings of the Sixth International Workshop on Semantic Evaluation*, pp. 265–274. Asso-
12378 ciation for Computational Linguistics.
- 12379 Morante, R. and W. Daelemans (2009). Learning the scope of hedge cues in biomedical
12380 texts. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language*
12381 *Processing*, pp. 28–36. Association for Computational Linguistics.

- 12382 Morante, R. and C. Sporleder (2012). Modality and negation: An introduction to the
 12383 special issue. *Computational linguistics* 38(2), 223–260.
- 12384 Mostafazadeh, N., A. Greish, N. Chambers, J. Allen, and L. Vanderwende (2016, June).
 12385 Caters: Causal and temporal relation scheme for semantic annotation of event struc-
 12386 tures. In *Proceedings of the Fourth Workshop on Events*, San Diego, California, pp. 51–61.
 12387 Association for Computational Linguistics.
- 12388 Mueller, T., H. Schmid, and H. Schütze (2013). Efficient higher-order CRFs for morpholog-
 12389 ical tagging. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
 12390 pp. 322–332.
- 12391 Müller, C. and M. Strube (2006). Multi-level annotation of linguistic data with mmax2.
 12392 *Corpus technology and language pedagogy: New resources, new tools, new methods* 3, 197–
 12393 214.
- 12394 Muralidharan, A. and M. A. Hearst (2013). Supporting exploratory text analysis in litera-
 12395 ture study. *Literary and linguistic computing* 28(2), 283–295.
- 12396 Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.
- 12397 Nakagawa, T., K. Inui, and S. Kurohashi (2010). Dependency tree-based sentiment classi-
 12398 fication using crfs with hidden variables. In *Proceedings of the North American Chapter of*
 12399 *the Association for Computational Linguistics (NAACL)*, pp. 786–794.
- 12400 Nakazawa, T., M. Yaguchi, K. Uchimoto, M. Utiyama, E. Sumita, S. Kurohashi, and H. Isa-
 12401 hara (2016). ASPEC: Asian scientific paper excerpt corpus. In *Proceedings of the Language*
 12402 *Resources and Evaluation Conference*, pp. 2204–2208.
- 12403 Navigli, R. (2009). Word sense disambiguation: A survey. *ACM Computing Surveys*
 12404 (*CSUR*) 41(2), 10.
- 12405 Neal, R. M. and G. E. Hinton (1998). A view of the em algorithm that justifies incremental,
 12406 sparse, and other variants. In *Learning in graphical models*, pp. 355–368. Springer.
- 12407 Nenkova, A. and K. McKeown (2012). A survey of text summarization techniques. In
 12408 *Mining text data*, pp. 43–76. Springer.
- 12409 Neubig, G. (2017). Neural machine translation and sequence-to-sequence models: A tu-
 12410 torial. *arXiv preprint arXiv:1703.01619*.
- 12411 Neubig, G., C. Dyer, Y. Goldberg, A. Matthews, W. Ammar, A. Anastasopoulos, M. Balles-
 12412 teros, D. Chiang, D. Clothiaux, T. Cohn, K. Duh, M. Faruqui, C. Gan, D. Garrette,
 12413 Y. Ji, L. Kong, A. Kuncoro, G. Kumar, C. Malaviya, P. Michel, Y. Oda, M. Richardson,
 12414 N. Saphra, S. Swayamdipta, and P. Yin (2017). Dynet: The dynamic neural network
 12415 toolkit.

- 12416 Neubig, G., Y. Goldberg, and C. Dyer (2017). On-the-fly operation batching in dynamic
12417 computation graphs. In *Neural Information Processing Systems (NIPS)*.
- 12418 Neubig, G., M. Sperber, X. Wang, M. Felix, A. Matthews, S. Padmanabhan, Y. Qi, D. S.
12419 Sachan, P. Arthur, P. Godard, J. Hewitt, R. Riad, and L. Wang (2018, March). XNMT:
12420 The extensible neural machine translation toolkit. In *Conference of the Association for
12421 Machine Translation in the Americas (AMTA) Open Source Software Showcase*, Boston.
- 12422 Neuhaus, P. and N. Bröker (1997). The complexity of recognition of linguistically adequate
12423 dependency grammars. In *eacl*, pp. 337–343.
- 12424 Newman, D., C. Chemudugunta, and P. Smyth (2006). Statistical entity-topic models. In
12425 *Proceedings of Knowledge Discovery and Data Mining (KDD)*, pp. 680–686.
- 12426 Ng, V. (2010). Supervised noun phrase coreference research: The first fifteen years. In
12427 *Proceedings of the 48th annual meeting of the association for computational linguistics*, pp.
12428 1396–1411. Association for Computational Linguistics.
- 12429 Nguyen, D. and A. S. Dogruöz (2013). Word level language identification in online multi-
12430 lingual communication. In *Proceedings of Empirical Methods for Natural Language Process-
12431 ing (EMNLP)*.
- 12432 Nguyen, D. T. and S. Joty (2017). A neural local coherence model. In *Proceedings of the
12433 Association for Computational Linguistics (ACL)*, pp. 1320–1330.
- 12434 Nigam, K., A. K. McCallum, S. Thrun, and T. Mitchell (2000). Text classification from
12435 labeled and unlabeled documents using em. *Machine learning* 39(2-3), 103–134.
- 12436 Nirenburg, S. and Y. Wilks (2001). What's in a symbol: ontology, representation and lan-
12437 guage. *Journal of Experimental & Theoretical Artificial Intelligence* 13(1), 9–23.
- 12438 Nivre, J. (2008). Algorithms for deterministic incremental dependency parsing. *Compu-
12439 tational Linguistics* 34(4), 513–553.
- 12440 Nivre, J., M.-C. de Marneffe, F. Ginter, Y. Goldberg, J. Hajič, C. D. Manning, R. McDonald,
12441 S. Petrov, S. Pyysalo, N. Silveira, R. Tsarfaty, and D. Zeman (2016, may). Universal de-
12442 pendencies v1: A multilingual treebank collection. In N. C. C. Chair), K. Choukri, T. De-
12443 clerck, S. Goggi, M. Grobelnik, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk,
12444 and S. Piperidis (Eds.), *Proceedings of the Tenth International Conference on Language Re-
12445 sources and Evaluation (LREC 2016)*, Paris, France. European Language Resources Asso-
12446 ciation (ELRA).
- 12447 Nivre, J. and J. Nilsson (2005). Pseudo-projective dependency parsing. In *Proceedings of the
12448 43rd Annual Meeting on Association for Computational Linguistics*, pp. 99–106. Association
12449 for Computational Linguistics.

- 12450 Novikoff, A. B. (1962). On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, Volume 12, pp. 615—622.
- 12451
- 12452 Och, F. J. and H. Ney (2003). A systematic comparison of various statistical alignment models. *Computational linguistics* 29(1), 19–51.
- 12453
- 12454 O'Connor, B., M. Krieger, and D. Ahn (2010). Tweetmotif: Exploratory search and topic summarization for twitter. In *Proceedings of the International Conference on Web and Social Media (ICWSM)*, pp. 384–385.
- 12455
- 12456
- 12457 Oflazer, K. and İ. Kuruöz (1994). Tagging and morphological disambiguation of turkish text. In *Proceedings of the fourth conference on Applied natural language processing*, pp. 144–149. Association for Computational Linguistics.
- 12458
- 12459
- 12460 Ohta, T., Y. Tateisi, and J.-D. Kim (2002). The genia corpus: An annotated research abstract corpus in molecular biology domain. In *Proceedings of the second international conference on Human Language Technology Research*, pp. 82–86. Morgan Kaufmann Publishers Inc.
- 12461
- 12462
- 12463 Onishi, T., H. Wang, M. Bansal, K. Gimpel, and D. McAllester (2016). Who did what: A large-scale person-centered cloze dataset. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 2230–2235.
- 12464
- 12465
- 12466 Owoputi, O., B. O'Connor, C. Dyer, K. Gimpel, N. Schneider, and N. A. Smith (2013). Improved part-of-speech tagging for online conversational text with word clusters. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 380–390.
- 12467
- 12468
- 12469
- 12470 Packard, W., E. M. Bender, J. Read, S. Oepen, and R. Dridan (2014). Simple negation scope resolution through deep parsing: A semantic solution to a semantic problem. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 69–78.
- 12471
- 12472
- 12473 Paice, C. D. (1990). Another stemmer. In *ACM SIGIR Forum*, Volume 24, pp. 56–61.
- 12474
- 12475 Pak, A. and P. Paroubek (2010). Twitter as a corpus for sentiment analysis and opinion mining. In *LREC*, Volume 10, pp. 1320–1326.
- 12476
- 12477 Palmer, F. R. (2001). *Mood and modality*. Cambridge University Press.
- 12478
- 12479 Palmer, M., D. Gildea, and P. Kingsbury (2005). The proposition bank: An annotated corpus of semantic roles. *Computational linguistics* 31(1), 71–106.
- 12480
- Pan, S. J. and Q. Yang (2010). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22(10), 1345–1359.

- 12481 Pan, X., T. Cassidy, U. Hermjakob, H. Ji, and K. Knight (2015). Unsupervised entity linking
12482 with abstract meaning representation. In *Proceedings of the North American Chapter of the*
12483 *Association for Computational Linguistics (NAACL)*, pp. 1130–1139.
- 12484 Pang, B. and L. Lee (2004). A sentimental education: Sentiment analysis using subjectivity
12485 summarization based on minimum cuts. In *Proceedings of the Association for Compu-*
12486 *tational Linguistics (ACL)*, pp. 271–278.
- 12487 Pang, B. and L. Lee (2005). Seeing stars: Exploiting class relationships for sentiment cate-
12488 gorization with respect to rating scales. In *Proceedings of the Association for Computational*
12489 *Linguistics (ACL)*, pp. 115–124.
- 12490 Pang, B. and L. Lee (2008). Opinion mining and sentiment analysis. *Foundations and trends*
12491 *in information retrieval* 2(1-2), 1–135.
- 12492 Pang, B., L. Lee, and S. Vaithyanathan (2002). Thumbs up?: sentiment classification using
12493 machine learning techniques. In *Proceedings of Empirical Methods for Natural Language*
12494 *Processing (EMNLP)*, pp. 79–86.
- 12495 Papineni, K., S. Roukos, T. Ward, and W.-J. Zhu (2002). Bleu: a method for automatic
12496 evaluation of machine translation. In *Proceedings of the Association for Computational*
12497 *Linguistics (ACL)*, pp. 311–318.
- 12498 Park, J. and C. Cardie (2012). Improving implicit discourse relation recognition through
12499 feature set optimization. In *Proceedings of the Special Interest Group on Discourse and Dia-*
12500 *logue (SIGDIAL)*, pp. 108–112.
- 12501 Parsons, T. (1990). *Events in the Semantics of English*, Volume 5. MIT Press.
- 12502 Pascanu, R., T. Mikolov, and Y. Bengio (2013). On the difficulty of training recurrent neural
12503 networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp.
12504 1310–1318.
- 12505 Paul, M., M. Federico, and S. Stüker (2010). Overview of the iwslt 2010 evaluation cam-
12506 paign. In *International Workshop on Spoken Language Translation (IWSLT) 2010*.
- 12507 Pedersen, T., S. Patwardhan, and J. Michelizzi (2004). Wordnet::similarity - measuring the
12508 relatedness of concepts. In *Proceedings of the North American Chapter of the Association for*
12509 *Computational Linguistics (NAACL)*, pp. 38–41.
- 12510 Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blon-
12511 del, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau,
12512 M. Brucher, M. Perrot, and E. Duchesnay (2011). Scikit-learn: Machine learning in
12513 Python. *Journal of Machine Learning Research* 12, 2825–2830.

- 12514 Pei, W., T. Ge, and B. Chang (2015). An effective neural network model for graph-based
 12515 dependency parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*,
 12516 pp. 313–322.
- 12517 Peldszus, A. and M. Stede (2013). From argument diagrams to argumentation mining
 12518 in texts: A survey. *International Journal of Cognitive Informatics and Natural Intelligence*
 12519 (*IJCINI*) 7(1), 1–31.
- 12520 Peldszus, A. and M. Stede (2015). An annotated corpus of argumentative microtexts. In
 12521 *Proceedings of the First Conference on Argumentation*.
- 12522 Peng, F., F. Feng, and A. McCallum (2004). Chinese segmentation and new word detec-
 12523 tion using conditional random fields. In *Proceedings of the International Conference on*
 12524 *Computational Linguistics (COLING)*, pp. 562.
- 12525 Pennington, J., R. Socher, and C. Manning (2014). Glove: Global vectors for word repre-
 12526 sentation. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
 12527 pp. 1532–1543.
- 12528 Pereira, F. and Y. Schabes (1992). Inside-outside reestimation from partially bracketed
 12529 corpora. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 128–
 12530 135.
- 12531 Pereira, F. C. N. and S. M. Shieber (2002). *Prolog and natural-language analysis*. Micromote
 12532 Publishing.
- 12533 Peters, M. E., M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer
 12534 (2018). Deep contextualized word representations. In *Proceedings of the North American*
 12535 *Chapter of the Association for Computational Linguistics (NAACL)*.
- 12536 Peterson, W. W., T. G. Birdsall, and W. C. Fox (1954). The theory of signal detectability.
 12537 *Transactions of the IRE professional group on information theory* 4(4), 171–212.
- 12538 Petrov, S., L. Barrett, R. Thibaux, and D. Klein (2006). Learning accurate, compact, and in-
 12539 terpretable tree annotation. In *Proceedings of the Association for Computational Linguistics*
 12540 (*ACL*).
- 12541 Petrov, S., D. Das, and R. McDonald (2012, May). A universal part-of-speech tagset. In
 12542 *Proceedings of LREC*.
- 12543 Petrov, S. and R. McDonald (2012). Overview of the 2012 shared task on parsing the web.
 12544 In *Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL)*,
 12545 Volume 59.
- 12546 Pinker, S. (2003). *The language instinct: How the mind creates language*. Penguin UK.

- 12547 Pinter, Y., R. Guthrie, and J. Eisenstein (2017). Mimicking word embeddings using
12548 subword RNNs. In *Proceedings of Empirical Methods for Natural Language Processing*
12549 (EMNLP).
- 12550 Pitler, E., A. Louis, and A. Nenkova (2009). Automatic sense prediction for implicit dis-
12551 course relations in text. In *Proceedings of the Association for Computational Linguistics*
12552 (ACL).
- 12553 Pitler, E. and A. Nenkova (2009). Using syntax to disambiguate explicit discourse con-
12554 nectives in text. In *Proceedings of the Association for Computational Linguistics* (ACL), pp.
12555 13–16.
- 12556 Pitler, E., M. Raghupathy, H. Mehta, A. Nenkova, A. Lee, and A. Joshi (2008). Easily iden-
12557 tifiable discourse relations. In *Proceedings of the International Conference on Computational*
12558 *Linguistics* (COLING), pp. 87–90.
- 12559 Plank, B., A. Søgaard, and Y. Goldberg (2016). Multilingual part-of-speech tagging with
12560 bidirectional long short-term memory models and auxiliary loss. In *Proceedings of the*
12561 *Association for Computational Linguistics* (ACL).
- 12562 Poesio, M., R. Stevenson, B. Di Eugenio, and J. Hitzeman (2004). Centering: A parametric
12563 theory and its instantiations. *Computational linguistics* 30(3), 309–363.
- 12564 Polanyi, L. and A. Zaenen (2006). Contextual valence shifters. In *Computing attitude and*
12565 *affect in text: Theory and applications*. Springer.
- 12566 Ponzetto, S. P. and M. Strube (2006). Exploiting semantic role labeling, wordnet and
12567 wikipedia for coreference resolution. In *Proceedings of the North American Chapter of*
12568 *the Association for Computational Linguistics* (NAACL), pp. 192–199.
- 12569 Ponzetto, S. P. and M. Strube (2007). Knowledge derived from wikipedia for computing
12570 semantic relatedness. *Journal of Artificial Intelligence Research* 30, 181–212.
- 12571 Poon, H. and P. Domingos (2008). Joint unsupervised coreference resolution with markov
12572 logic. In *Proceedings of Empirical Methods for Natural Language Processing* (EMNLP), pp.
12573 650–659.
- 12574 Poon, H. and P. Domingos (2009). Unsupervised semantic parsing. In *Proceedings of Em-
12575 pirical Methods for Natural Language Processing* (EMNLP), pp. 1–10.
- 12576 Popel, M., D. Marecek, J. Stepánek, D. Zeman, and Z. Zabokrtský (2013). Coordination
12577 structures in dependency treebanks. In *Proceedings of the Association for Computational*
12578 *Linguistics* (ACL), pp. 517–527.

- 12579 Popescu, A.-M., O. Etzioni, and H. Kautz (2003). Towards a theory of natural language
12580 interfaces to databases. In *Proceedings of Intelligent User Interfaces (IUI)*, pp. 149–157.
- 12581 Poplack, S. (1980). Sometimes i'll start a sentence in spanish y termino en español: toward
12582 a typology of code-switching1. *Linguistics* 18(7-8), 581–618.
- 12583 Porter, M. F. (1980). An algorithm for suffix stripping. *Program* 14(3), 130–137.
- 12584 Prabhakaran, V., O. Rambow, and M. Diab (2010). Automatic committed belief tagging.
12585 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.
12586 1014–1022.
- 12587 Pradhan, S., X. Luo, M. Recasens, E. Hovy, V. Ng, and M. Strube (2014). Scoring corefer-
12588 ence partitions of predicted mentions: A reference implementation. In *Proceedings of the*
12589 *Association for Computational Linguistics (ACL)*, pp. 30–35.
- 12590 Pradhan, S., L. Ramshaw, M. Marcus, M. Palmer, R. Weischedel, and N. Xue (2011).
12591 CoNLL-2011 shared task: Modeling unrestricted coreference in OntoNotes. In *Proceed-
12592 ings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*,
12593 pp. 1–27. Association for Computational Linguistics.
- 12594 Pradhan, S., W. Ward, K. Hacioglu, J. H. Martin, and D. Jurafsky (2005). Semantic role
12595 labeling using different syntactic views. In *Proceedings of the Association for Computational
12596 Linguistics (ACL)*, pp. 581–588.
- 12597 Prasad, R., N. Dinesh, A. Lee, E. Miltsakaki, L. Robaldo, A. Joshi, and B. Webber (2008).
12598 The Penn Discourse Treebank 2.0. In *Proceedings of LREC*.
- 12599 Punyakanok, V., D. Roth, and W.-t. Yih (2008). The importance of syntactic parsing and
12600 inference in semantic role labeling. *Computational Linguistics* 34(2), 257–287.
- 12601 Pustejovsky, J., P. Hanks, R. Sauri, A. See, R. Gaizauskas, A. Setzer, D. Radev, B. Sundheim,
12602 D. Day, L. Ferro, et al. (2003). The timebank corpus. In *Corpus linguistics*, Volume 2003,
12603 pp. 40. Lancaster, UK.
- 12604 Pustejovsky, J., B. Ingria, R. Sauri, J. Castano, J. Littman, R. Gaizauskas, A. Setzer, G. Katz,
12605 and I. Mani (2005). The specification language timeml. In *The language of time: A reader*,
12606 pp. 545–557. Oxford University Press.
- 12607 Qin, L., Z. Zhang, H. Zhao, Z. Hu, and E. Xing (2017). Adversarial connective-exploiting
12608 networks for implicit discourse relation classification. In *Proceedings of the Association
12609 for Computational Linguistics (ACL)*, pp. 1006–1017.
- 12610 Qiu, G., B. Liu, J. Bu, and C. Chen (2011). Opinion word expansion and target extraction
12611 through double propagation. *Computational linguistics* 37(1), 9–27.

- 12612 Quattoni, A., S. Wang, L.-P. Morency, M. Collins, and T. Darrell (2007). Hidden conditional
12613 random fields. *IEEE transactions on pattern analysis and machine intelligence* 29(10).
- 12614 Rahman, A. and V. Ng (2011). Narrowing the modeling gap: a cluster-ranking approach
12615 to coreference resolution. *Journal of Artificial Intelligence Research* 40, 469–521.
- 12616 Rajpurkar, P., J. Zhang, K. Lopyrev, and P. Liang (2016). Squad: 100,000+ questions for
12617 machine comprehension of text. In *Proceedings of Empirical Methods for Natural Language
12618 Processing (EMNLP)*, pp. 2383–2392.
- 12619 Ranzato, M., S. Chopra, M. Auli, and W. Zaremba (2016). Sequence level training with
12620 recurrent neural networks. In *Proceedings of the International Conference on Learning Rep-
12621 resentations (ICLR)*.
- 12622 Rao, D., D. Yarowsky, A. Shreevats, and M. Gupta (2010). Classifying latent user attributes
12623 in twitter. In *Proceedings of Workshop on Search and mining user-generated contents*.
- 12624 Ratinov, L. and D. Roth (2009). Design challenges and misconceptions in named entity
12625 recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language
12626 Learning*, pp. 147–155. Association for Computational Linguistics.
- 12627 Ratinov, L., D. Roth, D. Downey, and M. Anderson (2011). Local and global algorithms
12628 for disambiguation to wikipedia. In *Proceedings of the Association for Computational Lin-
12629 guistics (ACL)*, pp. 1375–1384.
- 12630 Ratliff, N. D., J. A. Bagnell, and M. Zinkevich (2007). (approximate) subgradient methods
12631 for structured prediction. In *Proceedings of Artificial Intelligence and Statistics (AISTATS)*,
12632 pp. 380–387.
- 12633 Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In *emnlp*,
12634 pp. 133–142.
- 12635 Ratnaparkhi, A., J. Reynar, and S. Roukos (1994). A maximum entropy model for preposi-
12636 tional phrase attachment. In *Proceedings of the workshop on Human Language Technology*,
12637 pp. 250–255. Association for Computational Linguistics.
- 12638 Read, J. (2005). Using emoticons to reduce dependency in machine learning techniques for
12639 sentiment classification. In *Proceedings of the ACL student research workshop*, pp. 43–48.
12640 Association for Computational Linguistics.
- 12641 Reisinger, D., R. Rudinger, F. Ferraro, C. Harman, K. Rawlins, and B. V. Durme (2015).
12642 Semantic proto-roles. *Transactions of the Association for Computational Linguistics* 3, 475–
12643 488.

- 12644 Reisinger, J. and R. J. Mooney (2010). Multi-prototype vector-space models of word mean-
 12645 ing. In *Proceedings of the North American Chapter of the Association for Computational Lin-*
 12646 *guistics (NAACL)*, pp. 109–117.
- 12647 Reiter, E. and R. Dale (2000). *Building natural language generation systems*. Cambridge
 12648 university press.
- 12649 Resnik, P., M. B. Olsen, and M. Diab (1999). The bible as a parallel corpus: Annotating the
 12650 ‘book of 2000 tongues’. *Computers and the Humanities* 33(1-2), 129–153.
- 12651 Resnik, P. and N. A. Smith (2003). The web as a parallel corpus. *Computational Linguis-*
 12652 *tics* 29(3), 349–380.
- 12653 Ribeiro, F. N., M. Araújo, P. Gonçalves, M. A. Gonçalves, and F. Benevenuto (2016).
 12654 Sentibench-a benchmark comparison of state-of-the-practice sentiment analysis meth-
 12655 ods. *EPJ Data Science* 5(1), 1–29.
- 12656 Richardson, M., C. J. Burges, and E. Renshaw (2013). MCTest: A challenge dataset for
 12657 the open-domain machine comprehension of text. In *Proceedings of Empirical Methods for*
 12658 *Natural Language Processing (EMNLP)*, pp. 193–203.
- 12659 Riedel, S., L. Yao, and A. McCallum (2010). Modeling relations and their mentions without
 12660 labeled text. In *Proceedings of the European Conference on Machine Learning and Principles*
 12661 *and Practice of Knowledge Discovery in Databases (ECML)*, pp. 148–163.
- 12662 Riedl, M. O. and R. M. Young (2010). Narrative planning: Balancing plot and character.
 12663 *Journal of Artificial Intelligence Research* 39, 217–268.
- 12664 Rieser, V. and O. Lemon (2011). *Reinforcement learning for adaptive dialogue systems: a data-*
 12665 *driven methodology for dialogue management and natural language generation*. Springer Sci-
 12666 *ence & Business Media*.
- 12667 Riloff, E. (1996). Automatically generating extraction patterns from untagged text. In
 12668 *Proceedings of the national conference on artificial intelligence*, pp. 1044–1049.
- 12669 Riloff, E. and J. Wiebe (2003). Learning extraction patterns for subjective expressions. In
 12670 *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pp.
 12671 105–112. Association for Computational Linguistics.
- 12672 Ritchie, G. (2001). Current directions in computational humour. *Artificial Intelligence Re-*
 12673 *view* 16(2), 119–135.
- 12674 Ritter, A., C. Cherry, and W. B. Dolan (2011). Data-driven response generation in social
 12675 media. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.
 12676 583–593.

- 12677 Ritter, A., S. Clark, Mausam, and O. Etzioni (2011). Named entity recognition in tweets:
12678 an experimental study. In *Proceedings of EMNLP*.
- 12679 Roark, B., M. Saraciar, and M. Collins (2007). Discriminative i_1/i_2 -gram language
12680 modeling. *Computer Speech & Language* 21(2), 373–392.
- 12681 Robert, C. and G. Casella (2013). *Monte Carlo statistical methods*. Springer Science & Busi-
12682 ness Media.
- 12683 Rosenfeld, R. (1996). A maximum entropy approach to adaptive statistical language mod-
12684 elling. *Computer Speech & Language* 10(3), 187–228.
- 12685 Ross, S., G. Gordon, and D. Bagnell (2011). A reduction of imitation learning and struc-
12686 tured prediction to no-regret online learning. In *Proceedings of Artificial Intelligence and*
12687 *Statistics (AISTATS)*, pp. 627–635.
- 12688 Roy, N., J. Pineau, and S. Thrun (2000). Spoken dialogue management using probabilistic
12689 reasoning. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 93–
12690 100.
- 12691 Rudnicky, A. and W. Xu (1999). An agenda-based dialog management architecture for
12692 spoken language systems. In *IEEE Automatic Speech Recognition and Understanding Work-
12693 shop*, Volume 13.
- 12694 Rush, A. M., S. Chopra, and J. Weston (2015). A neural attention model for abstractive sen-
12695 tence summarization. In *Proceedings of Empirical Methods for Natural Language Processing
(EMNLP)*, pp. 379–389.
- 12697 Rush, A. M., D. Sontag, M. Collins, and T. Jaakkola (2010). On dual decomposition and
12698 linear programming relaxations for natural language processing. In *Proceedings of Em-
12699 pirical Methods for Natural Language Processing (EMNLP)*, pp. 1–11.
- 12700 Russell, S. J. and P. Norvig (2009). *Artificial intelligence: a modern approach* (3rd ed.). Prentice
12701 Hall.
- 12702 Rutherford, A., V. Demberg, and N. Xue (2017). A systematic study of neural discourse
12703 models for implicit discourse relation. In *Proceedings of the European Chapter of the Asso-
12704 ciation for Computational Linguistics (EACL)*, pp. 281–291.
- 12705 Rutherford, A. T. and N. Xue (2014). Discovering implicit discourse relations through
12706 brown cluster pair representation and coreference patterns. In *Proceedings of the Euro-
12707 pean Chapter of the Association for Computational Linguistics (EACL)*.
- 12708 Sag, I. A., T. Baldwin, F. Bond, A. Copestate, and D. Flickinger (2002). Multiword expres-
12709 sions: A pain in the neck for nlp. In *International Conference on Intelligent Text Processing
and Computational Linguistics*, pp. 1–15. Springer.

- 12711 Sagae, K. (2009). Analysis of discourse structure with syntactic dependencies and data-
 12712 driven shift-reduce parsing. In *Proceedings of the 11th International Conference on Parsing
 Technologies*, pp. 81–84.
- 12714 Santos, C. D. and B. Zadrozny (2014). Learning character-level representations for part-of-
 12715 speech tagging. In *Proceedings of the International Conference on Machine Learning (ICML)*,
 12716 pp. 1818–1826.
- 12717 Sato, M.-A. and S. Ishii (2000). On-line em algorithm for the normalized gaussian network.
 12718 *Neural computation* 12(2), 407–432.
- 12719 Saurí, R. and J. Pustejovsky (2009). Factbank: a corpus annotated with event factuality.
 12720 *Language resources and evaluation* 43(3), 227.
- 12721 Saxe, A. M., J. L. McClelland, and S. Ganguli (2014). Exact solutions to the nonlinear
 12722 dynamics of learning in deep linear neural networks. In *Proceedings of the International
 Conference on Learning Representations (ICLR)*.
- 12724 Schank, R. C. and R. Abelson (1977). *Scripts, goals, plans, and understanding*. Hillsdale, NJ:
 12725 Erlbaum.
- 12726 Schapire, R. E. and Y. Singer (2000). Boostexter: A boosting-based system for text categori-
 12727 zation. *Machine learning* 39(2-3), 135–168.
- 12728 Schaul, T., S. Zhang, and Y. LeCun (2013). No more pesky learning rates. In *Proceedings of
 the International Conference on Machine Learning (ICML)*, pp. 343–351.
- 12730 Schnabel, T., I. Labutov, D. Mimno, and T. Joachims (2015). Evaluation methods for un-
 12731 supervised word embeddings. In *Proceedings of Empirical Methods for Natural Language
 Processing (EMNLP)*, pp. 298–307.
- 12733 Schneider, N., J. Flanigan, and T. O’Gorman (2015). The logic of amr: Practical, unified,
 12734 graph-based sentence semantics for nlp. In *Proceedings of the North American Chapter of
 the Association for Computational Linguistics (NAACL)*, pp. 4–5.
- 12736 Schütze, H. (1998). Automatic word sense discrimination. *Computational linguistics* 24(1),
 12737 97–123.
- 12738 Schwarm, S. E. and M. Ostendorf (2005). Reading level assessment using support vector
 12739 machines and statistical language models. In *Proceedings of the Association for Compu-
 tational Linguistics (ACL)*, pp. 523–530.
- 12741 See, A., P. J. Liu, and C. D. Manning (2017). Get to the point: Summarization with pointer-
 12742 generator networks. In *Proceedings of the Association for Computational Linguistics (ACL)*,
 12743 pp. 1073–1083.

- 12744 Sennrich, R., B. Haddow, and A. Birch (2016). Neural machine translation of rare words
12745 with subword units. In *Proceedings of the Association for Computational Linguistics (ACL)*,
12746 pp. 1715–1725.
- 12747 Serban, I. V., A. Sordoni, Y. Bengio, A. C. Courville, and J. Pineau (2016). Building end-to-
12748 end dialogue systems using generative hierarchical neural network models. In *Proceed-
12749 ings of the National Conference on Artificial Intelligence (AAAI)*, pp. 3776–3784.
- 12750 Settles, B. (2012). Active learning. *Synthesis Lectures on Artificial Intelligence and Machine
12751 Learning* 6(1), 1–114.
- 12752 Shang, L., Z. Lu, and H. Li (2015). Neural responding machine for short-text conversation.
12753 In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1577–1586.
- 12754 Shen, D. and M. Lapata (2007). Using semantic roles to improve question answering. In
12755 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 12–21.
- 12756 Shen, S., Y. Cheng, Z. He, W. He, H. Wu, M. Sun, and Y. Liu (2016). Minimum risk train-
12757 ing for neural machine translation. In *Proceedings of the Association for Computational
12758 Linguistics (ACL)*, pp. 1683–1692.
- 12759 Shen, W., J. Wang, and J. Han (2015). Entity linking with a knowledge base: Issues, tech-
12760 niques, and solutions. *IEEE Transactions on Knowledge and Data Engineering* 27(2), 443–
12761 460.
- 12762 Shieber, S. M. (1985). Evidence against the context-freeness of natural language. *Linguistics
12763 and Philosophy* 8(3), 333–343.
- 12764 Siegelmann, H. T. and E. D. Sontag (1995). On the computational power of neural nets.
12765 *Journal of computer and system sciences* 50(1), 132–150.
- 12766 Singh, S., A. Subramanya, F. Pereira, and A. McCallum (2011). Large-scale cross-
12767 document coreference using distributed inference and hierarchical models. In *Proceed-
12768 ings of the Association for Computational Linguistics (ACL)*, pp. 793–803.
- 12769 Sipser, M. (2012). *Introduction to the Theory of Computation*. Cengage Learning.
- 12770 Smith, D. A. and J. Eisner (2006). Minimum risk annealing for training log-linear models.
12771 In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 787–794.
- 12772 Smith, D. A. and J. Eisner (2008). Dependency parsing by belief propagation. In *Proceed-
12773 ings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 145–156.
- 12774 Smith, D. A. and N. A. Smith (2007). Probabilistic models of nonprojective dependency
12775 trees. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.
12776 132–140.

- 12777 Smith, N. A. (2011). Linguistic structure prediction. *Synthesis Lectures on Human Language
Technologies* 4(2), 1–274.
- 12779 Snover, M., B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul (2006). A study of transla-
12780 tion edit rate with targeted human annotation. In *Proceedings of association for machine
12781 translation in the Americas*, Volume 200.
- 12782 Snow, R., B. O'Connor, D. Jurafsky, and A. Y. Ng (2008). Cheap and fast—but is it good?:
12783 evaluating non-expert annotations for natural language tasks. In *Proceedings of Empirical
12784 Methods for Natural Language Processing (EMNLP)*, pp. 254–263.
- 12785 Snyder, B. and R. Barzilay (2007). Database-text alignment via structured multilabel classi-
12786 fication. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*,
12787 pp. 1713–1718.
- 12788 Socher, R., J. Bauer, C. D. Manning, and A. Y. Ng (2013). Parsing with compositional vector
12789 grammars. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 12790 Socher, R., B. Huval, C. D. Manning, and A. Y. Ng (2012). Semantic compositionality
12791 through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Em-
12792 pirical Methods in Natural Language Processing and Computational Natural Language Learn-
12793 ing*, pp. 1201–1211. Association for Computational Linguistics.
- 12794 Socher, R., A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts (2013).
12795 Recursive deep models for semantic compositionality over a sentiment treebank. In
12796 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12797 Søgaard, A. (2013). Semi-supervised learning and domain adaptation in natural language
12798 processing. *Synthesis Lectures on Human Language Technologies* 6(2), 1–103.
- 12799 Solorio, T. and Y. Liu (2008). Learning to predict code-switching points. In *Proceedings
12800 of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 973–981. Association
12801 for Computational Linguistics.
- 12802 Somasundaran, S., G. Namata, J. Wiebe, and L. Getoor (2009). Supervised and unsuper-
12803 vised methods in employing discourse relations for improving opinion polarity classi-
12804 fication. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12805 Somasundaran, S. and J. Wiebe (2009). Recognizing stances in online debates. In *Proceed-
12806 ings of the Association for Computational Linguistics (ACL)*, pp. 226–234.
- 12807 Song, L., B. Boots, S. M. Siddiqi, G. J. Gordon, and A. J. Smola (2010). Hilbert space
12808 embeddings of hidden markov models. In *Proceedings of the International Conference on
12809 Machine Learning (ICML)*, pp. 991–998.

- 12810 Song, L., Y. Zhang, X. Peng, Z. Wang, and D. Gildea (2016). Amr-to-text generation as
12811 a traveling salesman problem. In *Proceedings of Empirical Methods for Natural Language*
12812 *Processing (EMNLP)*, pp. 2084–2089.
- 12813 Soon, W. M., H. T. Ng, and D. C. Y. Lim (2001). A machine learning approach to corefer-
12814 ence resolution of noun phrases. *Computational linguistics* 27(4), 521–544.
- 12815 Sordoni, A., M. Galley, M. Auli, C. Brockett, Y. Ji, M. Mitchell, J.-Y. Nie, J. Gao, and B. Dolan
12816 (2015). A neural network approach to context-sensitive generation of conversational
12817 responses. In *Proceedings of the North American Chapter of the Association for Computational*
12818 *Linguistics (NAACL)*.
- 12819 Soriceut, R. and D. Marcu (2003). Sentence level discourse parsing using syntactic and
12820 lexical information. In *Proceedings of the North American Chapter of the Association for*
12821 *Computational Linguistics (NAACL)*, pp. 149–156.
- 12822 Sowa, J. F. (2000). *Knowledge representation: logical, philosophical, and computational founda-*
12823 *tions*. Pacific Grove, CA: Brooks/Cole.
- 12824 Spärck Jones, K. (1972). A statistical interpretation of term specificity and its application
12825 in retrieval. *Journal of documentation* 28(1), 11–21.
- 12826 Spitkovsky, V. I., H. Alshawi, D. Jurafsky, and C. D. Manning (2010). Viterbi training
12827 improves unsupervised dependency parsing. In *CONLL*, pp. 9–17.
- 12828 Sporleder, C. and M. Lapata (2005). Discourse chunking and its application to sen-
12829 tence compression. In *Proceedings of Empirical Methods for Natural Language Processing*
12830 (*EMNLP*), pp. 257–264.
- 12831 Sproat, R., A. Black, S. Chen, S. Kumar, M. Ostendorf, and C. Richards (2001). Normaliza-
12832 tion of non-standard words. *Computer Speech & Language* 15(3), 287–333.
- 12833 Sproat, R., W. Gale, C. Shih, and N. Chang (1996). A stochastic finite-state word-
12834 segmentation algorithm for chinese. *Computational linguistics* 22(3), 377–404.
- 12835 Sra, S., S. Nowozin, and S. J. Wright (2012). *Optimization for machine learning*. MIT Press.
- 12836 Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014).
12837 Dropout: A simple way to prevent neural networks from overfitting. *The Journal of*
12838 *Machine Learning Research* 15(1), 1929–1958.
- 12839 Srivastava, R. K., K. Greff, and J. Schmidhuber (2015). Training very deep networks. In
12840 *Neural Information Processing Systems (NIPS)*, pp. 2377–2385.

- 12841 Stab, C. and I. Gurevych (2014a). Annotating argument components and relations in per-
 12842 suasive essays. In *Proceedings of the International Conference on Computational Linguistics*
 12843 (*COLING*), pp. 1501–1510.
- 12844 Stab, C. and I. Gurevych (2014b). Identifying argumentative discourse structures in per-
 12845 suasive essays. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Lan-*
 12846 *guage Processing (EMNLP)*, pp. 46–56.
- 12847 Stede, M. (2011, nov). *Discourse Processing*, Volume 4 of *Synthesis Lectures on Human Lan-*
 12848 *guage Technologies*. Morgan & Claypool Publishers.
- 12849 Steedman, M. and J. Baldridge (2011). Combinatory categorial grammar. In *Non-*
 12850 *Transformational Syntax: Formal and Explicit Models of Grammar*. Wiley-Blackwell.
- 12851 Stenetorp, P., S. Pyysalo, G. Topić, T. Ohta, S. Ananiadou, and J. Tsujii (2012). Brat: a web-
 12852 based tool for nlp-assisted text annotation. In *Proceedings of the European Chapter of the*
 12853 *Association for Computational Linguistics (EACL)*, pp. 102–107.
- 12854 Stern, M., J. Andreas, and D. Klein (2017). A minimal span-based neural constituency
 12855 parser. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 12856 Stolcke, A., K. Ries, N. Coccaro, E. Shriberg, R. Bates, D. Jurafsky, P. Taylor, R. Martin,
 12857 C. Van Ess-Dykema, and M. Meteer (2000). Dialogue act modeling for automatic tag-
 12858 ging and recognition of conversational speech. *Computational linguistics* 26(3), 339–373.
- 12859 Stone, P. J. (1966). *The General Inquirer: A Computer Approach to Content Analysis*. The MIT
 12860 Press.
- 12861 Stoyanov, V., N. Gilbert, C. Cardie, and E. Riloff (2009). Conundrums in noun phrase
 12862 coreference resolution: Making sense of the state-of-the-art. In *Proceedings of the Associa-*
 12863 *tion for Computational Linguistics (ACL)*, pp. 656–664.
- 12864 Strang, G. (2016). *Introduction to linear algebra* (Fifth ed.). Wellesley, MA: Wellesley-
 12865 Cambridge Press.
- 12866 Strubell, E., P. Verga, D. Belanger, and A. McCallum (2017). Fast and accurate entity recog-
 12867 nition with iterated dilated convolutions. In *Proceedings of Empirical Methods for Natural*
 12868 *Language Processing (EMNLP)*.
- 12869 Suchanek, F. M., G. Kasneci, and G. Weikum (2007). Yago: a core of semantic knowledge.
 12870 In *Proceedings of the Conference on World-Wide Web (WWW)*, pp. 697–706.
- 12871 Sun, X., T. Matsuzaki, D. Okanohara, and J. Tsujii (2009). Latent variable perceptron algo-
 12872 rithm for structured classification. In *Proceedings of the International Joint Conference on*
 12873 *Artificial Intelligence (IJCAI)*, Volume 9, pp. 1236–1242.

- 12874 Sun, Y., L. Lin, D. Tang, N. Yang, Z. Ji, and X. Wang (2015). Modeling mention, context
12875 and entity with neural networks for entity disambiguation. In *IJCAI*, pp. 1333–1339.
- 12876 Sundermeyer, M., R. Schlüter, and H. Ney (2012). Lstm neural networks for language
12877 modeling. In *INTERSPEECH*.
- 12878 Surdeanu, M., J. Tibshirani, R. Nallapati, and C. D. Manning (2012). Multi-instance multi-
12879 label learning for relation extraction. In *Proceedings of Empirical Methods for Natural Lan-*
12880 *guage Processing (EMNLP)*, pp. 455–465.
- 12881 Sutskever, I., O. Vinyals, and Q. V. Le (2014). Sequence to sequence learning with neural
12882 networks. In *Neural Information Processing Systems (NIPS)*, pp. 3104–3112.
- 12883 Sutton, R. S. and A. G. Barto (1998). *Reinforcement learning: An introduction*, Volume 1. MIT
12884 press Cambridge.
- 12885 Sutton, R. S., D. A. McAllester, S. P. Singh, and Y. Mansour (2000). Policy gradient methods
12886 for reinforcement learning with function approximation. In *Neural Information Process-*
12887 *ing Systems (NIPS)*, pp. 1057–1063.
- 12888 Taboada, M., J. Brooke, M. Tofiloski, K. Voll, and M. Stede (2011). Lexicon-based methods
12889 for sentiment analysis. *Computational linguistics* 37(2), 267–307.
- 12890 Taboada, M. and W. C. Mann (2006). Rhetorical structure theory: Looking back and mov-
12891 ing ahead. *Discourse studies* 8(3), 423–459.
- 12892 Täckström, O., K. Ganchev, and D. Das (2015). Efficient inference and structured learning
12893 for semantic role labeling. *Transactions of the Association for Computational Linguistics* 3,
12894 29–41.
- 12895 Täckström, O., R. McDonald, and J. Uszkoreit (2012). Cross-lingual word clusters for
12896 direct transfer of linguistic structure. In *Proceedings of the North American Chapter of the*
12897 *Association for Computational Linguistics (NAACL)*, pp. 477–487.
- 12898 Tang, D., B. Qin, and T. Liu (2015). Document modeling with gated recurrent neural net-
12899 work for sentiment classification. In *Proceedings of Empirical Methods for Natural Language*
12900 *Processing (EMNLP)*, pp. 1422–1432.
- 12901 Taskar, B., C. Guestrin, and D. Koller (2003). Max-margin markov networks. In *Neural*
12902 *Information Processing Systems (NIPS)*.
- 12903 Tausczik, Y. R. and J. W. Pennebaker (2010). The psychological meaning of words: LIWC
12904 and computerized text analysis methods. *Journal of Language and Social Psychology* 29(1),
12905 24–54.

- 12906 Teh, Y. W. (2006). A hierarchical bayesian language model based on pitman-yor processes.
12907 In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 985–992.
- 12908 Tesnière, L. (1966). *Éléments de syntaxe structurale* (second ed.). Paris: Klincksieck.
- 12909 Teufel, S., J. Carletta, and M. Moens (1999). An annotation scheme for discourse-level
12910 argumentation in research articles. In *Proceedings of the European Chapter of the Association
12911 for Computational Linguistics (EACL)*, pp. 110–117.
- 12912 Teufel, S. and M. Moens (2002). Summarizing scientific articles: experiments with relevance
12913 and rhetorical status. *Computational linguistics* 28(4), 409–445.
- 12914 Thomas, M., B. Pang, and L. Lee (2006). Get out the vote: Determining support or opposition
12915 from Congressional floor-debate transcripts. In *Proceedings of Empirical Methods for
12916 Natural Language Processing (EMNLP)*, pp. 327–335.
- 12917 Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal
12918 Statistical Society. Series B (Methodological)*, 267–288.
- 12919 Titov, I. and J. Henderson (2007). Constituent parsing with incremental sigmoid belief
12920 networks. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 632–
12921 639.
- 12922 Toutanova, K., D. Klein, C. D. Manning, and Y. Singer (2003). Feature-rich part-of-speech
12923 tagging with a cyclic dependency network. In *Proceedings of the North American Chapter
12924 of the Association for Computational Linguistics (NAACL)*.
- 12925 Trivedi, R. and J. Eisenstein (2013). Discourse connectors for latent subjectivity in sentiment
12926 analysis. In *Proceedings of the North American Chapter of the Association for Computational
12927 Linguistics (NAACL)*, pp. 808–813.
- 12928 Tromble, R. W. and J. Eisner (2006). A fast finite-state relaxation method for enforcing
12929 global constraints on sequence decoding. In *Proceedings of the North American Chapter of
12930 the Association for Computational Linguistics (NAACL)*, pp. 423.
- 12931 Tsochantaridis, I., T. Hofmann, T. Joachims, and Y. Altun (2004). Support vector machine
12932 learning for interdependent and structured output spaces. In *Proceedings of the twenty-first
12933 international conference on Machine learning*, pp. 104. ACM.
- 12934 Tsvetkov, Y., M. Faruqui, W. Ling, G. Lample, and C. Dyer (2015). Evaluation of word
12935 vector representations by subspace alignment. In *Proceedings of Empirical Methods for
12936 Natural Language Processing (EMNLP)*, pp. 2049–2054.
- 12937 Tu, Z., Z. Lu, Y. Liu, X. Liu, and H. Li (2016). Modeling coverage for neural machine
12938 translation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 76–
12939 85.

- 12940 Turian, J., L. Ratinov, and Y. Bengio (2010). Word representations: a simple and general
12941 method for semi-supervised learning. In *Proceedings of the Association for Computational
12942 Linguistics (ACL)*, pp. 384–394.
- 12943 Turing, A. M. (2009). Computing machinery and intelligence. In *Parsing the Turing Test*,
12944 pp. 23–65. Springer.
- 12945 Turney, P. D. and P. Pantel (2010). From frequency to meaning: Vector space models of
12946 semantics. *Journal of Artificial Intelligence Research* 37, 141–188.
- 12947 Tutin, A. and R. Kittredge (1992). Lexical choice in context: generating procedural texts.
12948 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.
12949 763–769.
- 12950 Twain, M. (1997). *A Tramp Abroad*. New York: Penguin.
- 12951 Tzeng, E., J. Hoffman, T. Darrell, and K. Saenko (2015). Simultaneous deep transfer across
12952 domains and tasks. In *Proceedings of the IEEE International Conference on Computer Vision*,
12953 pp. 4068–4076.
- 12954 Usunier, N., D. Buffoni, and P. Gallinari (2009). Ranking with ordered weighted pairwise
12955 classification. In *Proceedings of the International Conference on Machine Learning (ICML)*,
12956 pp. 1057–1064.
- 12957 Uthus, D. C. and D. W. Aha (2013). The ubuntu chat corpus for multiparicipant chat
12958 analysis. In *AAAI Spring Symposium: Analyzing Microtext*, Volume 13, pp. 01.
- 12959 Utiyama, M. and H. Isahara (2001). A statistical model for domain-independent text seg-
12960 mentation. In *Proceedings of the 39th Annual Meeting on Association for Computational
12961 Linguistics*, pp. 499–506. Association for Computational Linguistics.
- 12962 Utiyama, M. and H. Isahara (2007). A comparison of pivot methods for phrase-based
12963 statistical machine translation. In *Human Language Technologies 2007: The Conference of
12964 the North American Chapter of the Association for Computational Linguistics; Proceedings of
12965 the Main Conference*, pp. 484–491.
- 12966 Uzuner, Ö., X. Zhang, and T. Sibanda (2009). Machine learning and rule-based approaches
12967 to assertion classification. *Journal of the American Medical Informatics Association* 16(1),
12968 109–115.
- 12969 Vadas, D. and J. R. Curran (2011). Parsing noun phrases in the penn treebank. *Computa-
12970 tional Linguistics* 37(4), 753–809.
- 12971 Van Eynde, F. (2006). NP-internal agreement and the structure of the noun phrase. *Journal
12972 of Linguistics* 42(1), 139–186.

- 12973 Van Gael, J., A. Vlachos, and Z. Ghahramani (2009). The infinite hmm for unsuper-
 12974 vised pos tagging. In *Proceedings of Empirical Methods for Natural Language Processing*
 12975 (EMNLP), pp. 678–687.
- 12976 Vaswani, A., S. Bengio, E. Brevdo, F. Chollet, A. N. Gomez, S. Gouws, L. Jones, L. Kaiser,
 12977 N. Kalchbrenner, N. Parmar, R. Sepassi, N. Shazeer, and J. Uszkoreit (2018). Ten-
 12978 sor2tensor for neural machine translation. *CoRR abs/1803.07416*.
- 12979 Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and
 12980 I. Polosukhin (2017). Attention is all you need. In *Neural Information Processing Systems*
 12981 (NIPS), pp. 6000–6010.
- 12982 Velldal, E., L. Øvreliid, J. Read, and S. Oopen (2012). Speculation and negation: Rules,
 12983 rankers, and the role of syntax. *Computational linguistics* 38(2), 369–410.
- 12984 Versley, Y. (2011). Towards finer-grained tagging of discourse connectives. In *Proceedings*
 12985 of the Workshop Beyound Semantics: Corpus-based Investigations of Pragmatic and Discourse
 12986 Phenomena, pp. 2–63.
- 12987 Vilain, M., J. Burger, J. Aberdeen, D. Connolly, and L. Hirschman (1995). A model-
 12988 theoretic coreference scoring scheme. In *Proceedings of the 6th conference on Message*
 12989 understanding, pp. 45–52. Association for Computational Linguistics.
- 12990 Vincent, P., H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol (2010). Stacked de-
 12991 noising autoencoders: Learning useful representations in a deep network with a local
 12992 denoising criterion. *Journal of Machine Learning Research* 11(Dec), 3371–3408.
- 12993 Vincze, V., G. Szarvas, R. Farkas, G. Móra, and J. Csirik (2008). The bioscope corpus:
 12994 biomedical texts annotated for uncertainty, negation and their scopes. *BMC bioinformat-
 12995 ics* 9(11), S9.
- 12996 Vinyals, O., A. Toshev, S. Bengio, and D. Erhan (2015). Show and tell: A neural image cap-
 12997 tion generator. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference*
 12998 on, pp. 3156–3164. IEEE.
- 12999 Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum
 13000 decoding algorithm. *IEEE transactions on Information Theory* 13(2), 260–269.
- 13001 Voll, K. and M. Taboada (2007). Not all words are created equal: Extracting semantic
 13002 orientation as a function of adjective relevance. In *Proceedings of Australian Conference*
 13003 on Artificial Intelligence.
- 13004 Wager, S., S. Wang, and P. S. Liang (2013). Dropout training as adaptive regularization. In
 13005 *Neural Information Processing Systems (NIPS)*, pp. 351–359.

- 13006 Wainwright, M. J. and M. I. Jordan (2008). Graphical models, exponential families, and
13007 variational inference. *Foundations and Trends® in Machine Learning* 1(1-2), 1–305.
- 13008 Walker, M. A. (2000). An application of reinforcement learning to dialogue strategy selec-
13009 tion in a spoken dialogue system for email. *Journal of Artificial Intelligence Research* 12,
13010 387–416.
- 13011 Walker, M. A., J. E. Cahn, and S. J. Whittaker (1997). Improvising linguistic style: Social
13012 and affective bases for agent personality. In *Proceedings of the first international conference*
13013 on *Autonomous agents*, pp. 96–105. ACM.
- 13014 Wang, C., N. Xue, and S. Pradhan (2015). A Transition-based Algorithm for AMR Parsing.
13015 In *Proceedings of the North American Chapter of the Association for Computational Linguistics*
13016 (NAACL), pp. 366–375.
- 13017 Wang, H., T. Onishi, K. Gimpel, and D. McAllester (2017). Emergent predication structure
13018 in hidden state vectors of neural readers. In *Proceedings of the 2nd Workshop on Repre-
13019 sentation Learning for NLP*, pp. 26–36.
- 13020 Weaver, W. (1955). Translation. *Machine translation of languages* 14, 15–23.
- 13021 Webber, B. (2004, sep). D-LTAG: extending lexicalized TAG to discourse. *Cognitive Sci-
13022 ence* 28(5), 751–779.
- 13023 Webber, B., M. Egg, and V. Kordoni (2012). Discourse structure and language technology.
13024 *Journal of Natural Language Engineering* 1.
- 13025 Webber, B. and A. Joshi (2012). Discourse structure and computation: past, present and
13026 future. In *Proceedings of the ACL-2012 Special Workshop on Rediscovering 50 Years of Dis-
13027 coveries*, pp. 42–54. Association for Computational Linguistics.
- 13028 Wei, G. C. and M. A. Tanner (1990). A monte carlo implementation of the em algorithm
13029 and the poor man’s data augmentation algorithms. *Journal of the American Statistical
13030 Association* 85(411), 699–704.
- 13031 Weinberger, K., A. Dasgupta, J. Langford, A. Smola, and J. Attenberg (2009). Feature
13032 hashing for large scale multitask learning. In *Proceedings of the International Conference
13033 on Machine Learning (ICML)*, pp. 1113–1120.
- 13034 Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language
13035 communication between man and machine. *Communications of the ACM* 9(1), 36–45.
- 13036 Wellner, B. and J. Pustejovsky (2007). Automatically identifying the arguments of dis-
13037 course connectives. In *Proceedings of Empirical Methods for Natural Language Processing
13038 (EMNLP)*, pp. 92–101.

- 13039 Wen, T.-H., M. Gasic, N. Mrkšić, P.-H. Su, D. Vandyke, and S. Young (2015). Semantically
13040 conditioned lstm-based natural language generation for spoken dialogue systems. In
13041 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1711–1721.
- 13042 Weston, J., S. Bengio, and N. Usunier (2011). Wsabie: Scaling up to large vocabulary image
13043 annotation. In *IJCAI*, Volume 11, pp. 2764–2770.
- 13044 Wiebe, J., T. Wilson, and C. Cardie (2005). Annotating expressions of opinions and emo-
13045 tions in language. *Language resources and evaluation* 39(2), 165–210.
- 13046 Wieting, J., M. Bansal, K. Gimpel, and K. Livescu (2015). Towards universal paraphrastic
13047 sentence embeddings. *arXiv preprint arXiv:1511.08198*.
- 13048 Wieting, J., M. Bansal, K. Gimpel, and K. Livescu (2016). CHARAGRAM: Embedding
13049 words and sentences via character n-grams. In *Proceedings of Empirical Methods for Nat-*
13050 *ural Language Processing (EMNLP)*, pp. 1504–1515.
- 13051 Williams, J. D. and S. Young (2007). Partially observable markov decision processes for
13052 spoken dialog systems. *Computer Speech & Language* 21(2), 393–422.
- 13053 Williams, P., R. Sennrich, M. Post, and P. Koehn (2016). Syntax-based statistical machine
13054 translation. *Synthesis Lectures on Human Language Technologies* 9(4), 1–208.
- 13055 Wilson, T., J. Wiebe, and P. Hoffmann (2005). Recognizing contextual polarity in phrase-
13056 level sentiment analysis. In *Proceedings of Empirical Methods for Natural Language Pro-*
13057 *cessing (EMNLP)*, pp. 347–354.
- 13058 Winograd, T. (1972). Understanding natural language. *Cognitive psychology* 3(1), 1–191.
- 13059 Wiseman, S., A. M. Rush, and S. M. Shieber (2016). Learning global features for corefer-
13060 ence resolution. In *Proceedings of the North American Chapter of the Association for Compu-*
13061 *tational Linguistics (NAACL)*, pp. 994–1004.
- 13062 Wiseman, S., S. Shieber, and A. Rush (2017). Challenges in data-to-document generation.
13063 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 2253–
13064 2263.
- 13065 Wiseman, S. J., A. M. Rush, S. M. Shieber, and J. Weston (2015). Learning anaphoricity and
13066 antecedent ranking features for coreference resolution. In *Proceedings of the Association*
13067 *for Computational Linguistics (ACL)*.
- 13068 Wolf, F. and E. Gibson (2005). Representing discourse coherence: A corpus-based study.
13069 *Computational Linguistics* 31(2), 249–287.
- 13070 Wolfe, T., M. Dredze, and B. Van Durme (2017). Pocket knowledge base population. In
13071 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 305–310.

- 13072 Wong, Y. W. and R. Mooney (2007). Generation by inverting a semantic parser that uses
13073 statistical machine translation. In *Proceedings of the North American Chapter of the Association
13074 for Computational Linguistics (NAACL)*, pp. 172–179.
- 13075 Wong, Y. W. and R. J. Mooney (2006). Learning for semantic parsing with statistical ma-
13076 chine translation. In *Proceedings of the North American Chapter of the Association for Com-
13077 putational Linguistics (NAACL)*, pp. 439–446.
- 13078 Wu, B. Y. and K.-M. Chao (2004). *Spanning trees and optimization problems*. CRC Press.
- 13079 Wu, D. (1997). Stochastic inversion transduction grammars and bilingual parsing of par-
13080 allel corpora. *Computational linguistics* 23(3), 377–403.
- 13081 Wu, F. and D. S. Weld (2010). Open information extraction using wikipedia. In *Proceedings
13082 of the Association for Computational Linguistics (ACL)*, pp. 118–127.
- 13083 Wu, X., R. Ward, and L. Bottou (2018). Wngrad: Learn the learning rate in gradient de-
13084 scent. *arXiv preprint arXiv:1803.02865*.
- 13085 Wu, Y., M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao,
13086 Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws,
13087 Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young,
13088 J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean (2016).
13089 Google’s neural machine translation system: Bridging the gap between human and ma-
13090 chine translation. *CoRR abs/1609.08144*.
- 13091 Xia, F. (2000). The part-of-speech tagging guidelines for the penn chinese treebank (3.0).
13092 Technical report, University of Pennsylvania Institute for Research in Cognitive Science.
- 13093 Xu, K., J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio
13094 (2015). Show, attend and tell: Neural image caption generation with visual attention.
13095 In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 2048–2057.
- 13096 Xu, W., X. Liu, and Y. Gong (2003). Document clustering based on non-negative matrix
13097 factorization. In *SIGIR*, pp. 267–273. ACM.
- 13098 Xu, Y., L. Mou, G. Li, Y. Chen, H. Peng, and Z. Jin (2015). Classifying relations via long
13099 short term memory networks along shortest dependency paths. In *Proceedings of Empir-
13100 ical Methods for Natural Language Processing (EMNLP)*, pp. 1785–1794.
- 13101 Xuan Bach, N., N. L. Minh, and A. Shimazu (2012). A reranking model for discourse seg-
13102 mentation using subtree features. In *Proceedings of the Special Interest Group on Discourse
13103 and Dialogue (SIGDIAL)*.

- 13104 Xue, N. et al. (2003). Chinese word segmentation as character tagging. *Computational
13105 Linguistics and Chinese Language Processing* 8(1), 29–48.
- 13106 Xue, N., H. T. Ng, S. Pradhan, R. Prasad, C. Bryant, and A. T. Rutherford (2015). The
13107 CoNLL-2015 shared task on shallow discourse parsing. In *Proceedings of the Conference
13108 on Natural Language Learning (CoNLL)*.
- 13109 Xue, N., H. T. Ng, S. Pradhan, A. Rutherford, B. L. Webber, C. Wang, and H. Wang (2016).
13110 Conll 2016 shared task on multilingual shallow discourse parsing. In *CoNLL Shared
13111 Task*, pp. 1–19.
- 13112 Yamada, H. and Y. Matsumoto (2003). Statistical dependency analysis with support vector
13113 machines. In *Proceedings of IWPT*, Volume 3, pp. 195–206.
- 13114 Yamada, K. and K. Knight (2001). A syntax-based statistical translation model. In *Proceed-
13115 ings of the 39th Annual Meeting on Association for Computational Linguistics*, pp. 523–530.
13116 Association for Computational Linguistics.
- 13117 Yang, B. and C. Cardie (2014). Context-aware learning for sentence-level sentiment anal-
13118 ysis with posterior regularization. In *Proceedings of the Association for Computational Lin-
13119 guistics (ACL)*.
- 13120 Yang, Y., M.-W. Chang, and J. Eisenstein (2016). Toward socially-infused information ex-
13121 traction: Embedding authors, mentions, and entities. In *Proceedings of Empirical Methods
13122 for Natural Language Processing (EMNLP)*.
- 13123 Yang, Y. and J. Eisenstein (2013). A log-linear model for unsupervised text normalization.
13124 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 13125 Yang, Y. and J. Eisenstein (2015). Unsupervised multi-domain adaptation with feature em-
13126 beddings. In *Proceedings of the North American Chapter of the Association for Computational
13127 Linguistics (NAACL)*.
- 13128 Yang, Y., W.-t. Yih, and C. Meek (2015). WikiQA: A challenge dataset for open-domain
13129 question answering. In *Proceedings of Empirical Methods for Natural Language Processing
13130 (EMNLP)*, pp. 2013–2018.
- 13131 Yannakoudakis, H., T. Briscoe, and B. Medlock (2011). A new dataset and method for
13132 automatically grading esol texts. In *Proceedings of the 49th Annual Meeting of the Associa-
13133 tion for Computational Linguistics: Human Language Technologies-Volume 1*, pp. 180–189.
13134 Association for Computational Linguistics.
- 13135 Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised meth-
13136 ods. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 189–196.
13137 Association for Computational Linguistics.

- 13138 Yee, L. C. and T. Y. Jones (2012, March). Apple ceo in china mission to clear up problems.
13139 *Reuters*. retrieved on March 26, 2017.
- 13140 Yi, Y., C.-Y. Lai, S. Petrov, and K. Keutzer (2011, October). Efficient parallel cky parsing on
13141 gpus. In *Proceedings of the 12th International Conference on Parsing Technologies*, Dublin,
13142 Ireland, pp. 175–185. Association for Computational Linguistics.
- 13143 Yu, C.-N. J. and T. Joachims (2009). Learning structural svms with latent variables. In
13144 *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1169–1176.
- 13145 Yu, F. and V. Koltun (2016). Multi-scale context aggregation by dilated convolutions. In
13146 *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 13147 Zaidan, O. F. and C. Callison-Burch (2011). Crowdsourcing translation: Professional qual-
13148 ity from non-professionals. In *Proceedings of the Association for Computational Linguistics*
13149 (*ACL*), pp. 1220–1229.
- 13150 Zaremba, W., I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv*
13151 preprint *arXiv:1409.2329*.
- 13152 Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint*
13153 *arXiv:1212.5701*.
- 13154 Zelenko, D., C. Aone, and A. Richardella (2003). Kernel methods for relation extraction.
13155 *The Journal of Machine Learning Research* 3, 1083–1106.
- 13156 Zelle, J. M. and R. J. Mooney (1996). Learning to parse database queries using induc-
13157 tive logic programming. In *Proceedings of the National Conference on Artificial Intelligence*
13158 (*AAAI*), pp. 1050–1055.
- 13159 Zeng, D., K. Liu, S. Lai, G. Zhou, and J. Zhao (2014). Relation classification via convolu-
13160 tional deep neural network. In *Proceedings of the International Conference on Computational*
13161 *Linguistics (COLING)*, pp. 2335–2344.
- 13162 Zettlemoyer, L. S. and M. Collins (2005). Learning to map sentences to logical form: Struc-
13163 tured classification with probabilistic categorial grammars. In *Proceedings of UAI*.
- 13164 Zhang, X., J. Zhao, and Y. LeCun (2015). Character-level convolutional networks for text
13165 classification. In *Neural Information Processing Systems (NIPS)*, pp. 649–657.
- 13166 Zhang, Y. and S. Clark (2008). A tale of two parsers: investigating and combining graph-
13167 based and transition-based dependency parsing using beam-search. In *Proceedings of*
13168 *Empirical Methods for Natural Language Processing (EMNLP)*, pp. 562–571.

- 13169 Zhang, Y., T. Lei, R. Barzilay, T. Jaakkola, and A. Globerson (2014). Steps to excellence:
13170 Simple inference with refined scoring of dependency trees. In *Proceedings of the Association
13171 for Computational Linguistics (ACL)*, pp. 197–207.
- 13172 Zhang, Y. and J. Nivre (2011). Transition-based dependency parsing with rich non-local
13173 features. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 188–193.
- 13174 Zhang, Z. (2017). A note on counting dependency trees. *arXiv preprint arXiv:1708.08789*.
- 13175 Zhou, J. and W. Xu (2015). End-to-end learning of semantic role labeling using recurrent
13176 neural networks. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
13177 1127–1137.
- 13178 Zhu, J., Z. Nie, X. Liu, B. Zhang, and J.-R. Wen (2009). Statsnowball: a statistical approach
13179 to extracting entity relationships. In *Proceedings of the Conference on World-Wide Web
13180 (WWW)*, pp. 101–110.
- 13181 Zhu, X., Z. Ghahramani, and J. D. Lafferty (2003). Semi-supervised learning using gaus-
13182 sian fields and harmonic functions. In *Proceedings of the International Conference on Ma-
13183 chine Learning (ICML)*, pp. 912–919.
- 13184 Zhu, X. and A. B. Goldberg (2009). Introduction to semi-supervised learning. *Synthesis
13185 lectures on artificial intelligence and machine learning* 3(1), 1–130.
- 13186 Zipf, G. K. (1949). Human behavior and the principle of least effort.
- 13187 Zirn, C., M. Niepert, H. Stuckenschmidt, and M. Strube (2011). Fine-grained sentiment
13188 analysis with structural features. In *IJCNLP*, Chiang Mai, Thailand, pp. 336–344.
- 13189 Zou, W. Y., R. Socher, D. Cer, and C. D. Manning (2013). Bilingual word embeddings
13190 for phrase-based machine translation. In *Proceedings of Empirical Methods for Natural
13191 Language Processing (EMNLP)*, pp. 1393–1398.

13192 **Index**

- 13193 *K*-means, 112
13194 soft, 113
13195 α -conversion, 310
13196 β -conversion, 307
13197 β -reduction, 307
13198 *n*-gram
 language model, 214
13199 *n*-gram, 40
13200 *F*-MEASURE, 98
 balanced, 99
13203 macro, 98
13204 micro, 99
13205 BLEU, 455
13206 WordNet, 90

13207 ablation test, 100
13208 Abstract Meaning Representation (AMR), 323, 335
13209 abstractive summarization, 413, 486
13211 accuracy, 38, 97
13212 action (reinforcement learning), 385
13213 AdaDelta (online optimization), 77
13214 AdaGrad, 55
13215 AdaGrad (online optimization), 77
13216 Adam (online optimization), 77
13217 adequacy (translation), 455
13218 adjectives, 193
13219 adjuncts (semantics), 322, 326
13220 adpositions, 194
13221 adverbs, 193
13222 adversarial networks, 129

13223 affix (morphology), 211
13224 agenda-based dialogue systems, 490
13225 agenda-based parsing, 293
13226 agent (thematic role), 324
13227 alignment, 337
 alignment (machine translation), 454, 459
13230 alignment (text generation), 481
13231 Amazon Mechanical Turk, 107
13232 ambiguity, 226, 234
 derivational, 238
 spurious, 238
13235 ambiguity, attachment, 245
13236 ambiguity, complement structure, 245
13237 ambiguity, coordination scope, 245
13238 ambiguity, modifier scope, 245
13239 ambiguity, particle versus preposition, 245
13240 anaphoric, 380
13242 anchored productions, 248
13243 animacy (semantics), 323
13244 annealing, 474
13245 antecedent (coreference), 371, 379
13246 antonymy, 90, 295
13247 apophony, 210
13248 arc-eager dependency parsing, 283, 284
13249 arc-factored dependency parsing, 277
13250 arc-standard dependency parsing, 283
13251 area under the curve (AUC), 99
13252 argumentation, 412
 mining, 413

- 13254 arguments, 423
 13255 article (syntax), 198
 13256 aspect, 193
 13257 aspect-based opinion mining, 87
 13258 attachment ambiguity, 271
 13259 attention mechanism, 391, 446, 466, 482
 13260 autoencoder, 365
 13261 autoencoder, denoising, 365, 428
 13262 automated theorem provers, 302
 13263 automatic differentiation, 72
 13264 auxiliary verbs, 194
 13265 average mutual information, 353
 13266 averaged perceptron, 42
- 13267 backchannel, 203
 13268 backoff, 146
 13269 Katz, 146
 13270 backpropagation, 71
 through time, 152
 13272 backward recurrence, 181, 182
 13273 backward-looking center, 402
 13274 bag of words, 29
 13275 balanced test set, 97
 13276 batch normalization, 76
 13277 Baum-Welch algorithm, 186
 13278 Bayes' rule, 500
 13279 Bayesian nonparametrics, 119, 264
 13280 beam sampling, 188
 13281 beam search, 286, 384, 471, 472
 13282 Bell number, 382
 13283 bias, 38
 13284 bias (learning theory), 143
 13285 bias-variance tradeoff, 38, 145
 13286 bidirectional LSTM, 467
 13287 bidirectional recurrent neural network,
 184
 13289 bigrams, 40, 86
 13290 bilexical features, 280
 13291 bilinear product, 350
 13292 binarization (context-free grammar), 244
 13293 binarization (of context-free grammars),
 227
 13295 binomial distribution, 101
 13296 binomial random variable, 504
 13297 binomial test, 100
 13298 BIO notation, 200, 333
 13299 biomedical natural language processing,
 199
 13301 bipartite graph, 339
 13302 Bonferroni correction, 103
 13303 boolean semiring, 215
 13304 boosting, 64
 13305 bootstrap samples, 102
 13306 brevity penalty (machine translation),
 456
 13308 Brown clusters, 347, 352
 13309 byte-pair encoding, 363, 470
- 13310 c-command, 373
 13311 case marking, 198, 235
 13312 Catalan number, 241
 13313 cataphora, 372
 13314 center embedding, 223
 13315 centering theory, 375, 402
 13316 chain rule of probability, 499
 13317 chance agreement, 106
 13318 character-level language models, 157
 13319 chart parsing, 242
 13320 chatbots, 492
 13321 Chomsky Normal Form (CNF), 227
 13322 Chu-Liu-Edmonds algorithm, 278
 13323 CKY algorithm, 242
 13324 class imbalance (in machine learning), 97
 13325 classification, 29
 13326 large margin, 46
 13327 lexicon-based, 88
 13328 weights, 29
 13329 closed-vocabulary, 157
 13330 closure (of classes of formal languages),
 208
 13331 cloze question answering, 445

- 13333 cluster ranking, 383
 13334 clustering, 112
 13335 co-hypernymy, 295
 13336 co-training, 124
 13337 coarse-to-fine attention, 484
 13338 code switching, 196, 202
 13339 Cohen's Kappa, 106
 13340 coherence, 416
 13341 cohesion, 399, 420
 13342 collapsed Gibbs sampling, 132
 13343 collective entity linking, 428, 429
 13344 collocation extraction, 364
 13345 combinatorial optimization, 19
 13346 combinatory categorial grammar, 236
 13347 complement clause, 229
 13348 complement event (probability), 498
 13349 composition (CCG), 237
 13350 compositional vector grammars, 411
 13351 compositionality, 18, 21, 362
 13352 computation graph, 64, 71
 13353 computational linguistics (versus
 natural language processing), 13
 13355 computational social science, 17
 13356 concept (AMR), 335
 13357 conditional independence, 170, 502
 13358 conditional random field, 178
 13359 confidence interval, 102
 13360 configuration (transition-based parsing),
 283
 13362 connected (graph theory), 337
 13363 consistency (logic), 305
 13364 constants (logic), 300
 13365 constituents, 228
 13366 constrained optimization, 331
 13367 constraints, 331
 13368 content selection (text generation), 479
 13369 content words, 194
 13370 context vector (attentional neural
 translation), 468
 13372 context-free grammars, 224
 13373 probabilistic (PCFGs), 251
 13374 weighted, 234, 248
 13375 context-free languages, 223, 224
 13376 context-free production, 225
 13377 recursive, 225
 13378 unary, 226
 13379 context-sensitive languages, 234
 13380 continuous bag-of-words (CBOW), 355
 13381 contradiction, 366
 13382 conversational turns, 203
 13383 convexity, 44, 74, 315, 504, 507
 13384 biconvexity, 118
 13385 convolution
 13386 dilated, 79, 201
 13387 narrow, 79
 13388 one-dimensional, 79
 13389 wide, 79
 13390 convolutional neural networks, 186, 435
 13391 cooperative principle, 371
 13392 coordinate ascent, 118
 13393 coordinating conjunctions, 194
 13394 coordinating discourse relations, 409
 13395 copula, 193, 233, 274
 13396 coreference resolution, 367, 371
 13397 cross-document, 426
 13398 coreferent, 371
 13399 cosine similarity, 359, 400
 13400 cost-augmented decoding, 49, 177
 13401 coverage (summarization), 413
 13402 coverage loss, 487
 13403 critical point, 74, 507
 13404 cross-entropy, 69, 436
 13405 cross-serial dependencies, 235
 13406 cross-validation, 39
 13407 crowdsourcing, 107
 13408 cumulative probability distribution, 102
 13409 decidability (logic), 305
 13410 decision trees, 64
 13411 deep LSTM, 465
 13412 definiteness, 199
 13413 delta function, *see* indicator function

- 13414 denotation (semantics), 300
 13415 dependency path, 91
 13416 dependency grammar, 271
 13418 dependency graph, 272
 13419 dependency parse, 271
 13420 dependency path, 330, 433
 13421 dependent, 272
 13422 derivation (context-free languages), 225
 13423 derivation (parsing), 283, 288
 13424 derivation (semantic parsing), 308, 312
 13425 determiner, 195
 13426 determiner phrase, 231
 13427 development set, 39, 97
 13428 dialogue acts, 106, 203, 493
 13429 dialogue management, 490
 13430 dialogue systems, 141, 489
 13431 digital humanities, 17, 85
 13432 Dirichlet distribution, 131
 13433 discounting, 146
 absolute, 146
 13435 discourse, 399
 13436 discourse connectives, 405
 13437 discourse depth, 414
 13438 discourse depth tree, 415
 13439 discourse parsing, 404
 13440 discourse relations, 367, 404
 13441 discourse segment, 399
 13442 discourse sense classification, 406
 13443 discourse unit, 409
 13444 discrete random variable, 502
 13445 disjoint events (probability), 498
 13446 distant supervision, 134, 438, 439
 13447 distributed semantics, 346
 13448 distributional
 semantics, 22
 statistics, 91
 13451 distributional hypothesis, 345, 346
 13452 distributional semantics, 346
 13453 distributional statistics, 263, 346
 13454 document frequency, 427
 13455 domain adaptation, 111, 127
 by projection, 128
 13457 dropout, 73, 153
 13458 dual decomposition, 333
 13459 dynamic computation graphs, 72
 13460 dynamic oracle, 291
 13461 dynamic programming, 165
 13462 dynamic semantics, 317, 404
 13463 E-step, 115
 13464 early stopping, 43, 77
 13465 early update, 290
 13466 easy-first parsing, 293
 13467 edit distance, 217, 457
 13468 effective counts, 145
 13469 elementary discourse units, 409
 13470 elementwise nonlinearity, 66
 13471 Elman unit, 151
 13472 ELMo (embeddings from language models), 361
 13473 embedding, 183, 355
 13475 emotion, 88
 13476 empirical Bayes, 132
 13477 empty string, 208
 13478 encoder-decoder, 464
 13479 encoder-decoder model, 365, 482
 13480 ensemble, 334
 13481 ensemble learning, 466
 13482 ensemble methods, 64
 13483 entailment, 305, 366
 13484 entities, 423
 13485 entity embeddings, 427
 13486 entity grid, 403
 13487 entity linking, 371, 423, 425, 436
 13488 entropy, 57, 115
 13489 estimation, 504
 13490 EuroParl corpus, 457
 13491 evaluation
 extrinsic, 155
 intrinsic, 155
 13494 event, 441

- 13495 event (probability), 497
 13496 event coreference, 441
 13497 event detection, 441
 13498 event semantics, 321
 13499 events, 423
 13500 evidentiality, 198, 443
 13501 exchange clustering, 354
 13502 expectation, 503
 13503 expectation-maximization, 113
 hard, 118
 13505 in language modeling, 147
 13506 in machine translation, 461
 13507 incremental, 118
 13508 online, 118
 13509 explicit semantic analysis, 348
 13510 extra-propositional semantics, 443
 13511 extractive question-answering, 446
 13512 extractive summarization, 413
 13513 extrinsic evaluation, 359
- 13514 factoid questions, 338
 13515 factoids, 444
 13516 factor graph, 179
 13517 factuality, 443
 13518 false discovery rate, 103
 13519 false negative, 97
 13520 false positive, 97, 501
 rate, 99
 13522 false positive rate, 500
 13523 feature
 co-adaptation, 74
 13525 function, 30, 39
 13526 hashing, 96
 13527 noising, 74
 13528 selection, 56
 13529 features, 18
 13530 collocation, 91
 13531 emission, 164
 13532 lexical, 63
 13533 transition, 164
 13534 fine-tuned word embeddings, 361
- 13535 finite state
 13536 acceptor, 209
 13537 acceptor, chain, 221
 13538 acceptors, weighted, 213
 13539 automata, 209
 13540 automaton, deterministic, 210
 13541 composition, 220
 13542 transducers, 212, 217
 13543 first-order logic, 303
 13544 fluency, 141
 13545 fluency (translation), 455
 13546 focus, 336
 13547 formal language theory, 207
 13548 forward
 recurrence, 180
 13550 variable, 180, 182
 13551 forward-backward algorithm, 181, 222,
 255
 13553 forward-looking centers, 402
 13554 frame, 489
 13555 frame elements, 326
 13556 FrameNet, 326
 13557 frames, 326
 13558 Frobenius norm, 121
 13559 function (first-order logic), 304
 13560 function words, 194
 13561 functional segmentation, 399, 401
- 13562 garden path sentence, 162
 13563 gate (neural networks), 68, 467
 13564 gazetteer, 200, 377, 433
 13565 generalization, 43
 13566 generalized linear models, 57
 13567 generative model, 33
 for coreference, 383
 for parsing, 251
 13570 generative process, 147
 13571 generic referents, 376
 13572 Gibbs sampling, 131, 429
 13573 gloss, 141, 195, 455
 13574 government and binding theory, 373

- 13575 gradient, 45
 13576 clipping, 76
 13577 descent, 53
 13578 exploding, 153
 13579 vanishing, 67, 153
 13580 Gram matrix, 434
 13581 grammar equivalence
 strong, 226
 weak, 226
 13584 grammar induction, 257
 13585 grammaticality, 416
 13586 graph-based dependency parsing, 277
 13587 graphical model, 170
 13588 graphics processing units (GPUs), 186,
 201
 13590 grid search, 38
 13591 Hamming cost, 177
 13592 Hansards corpus, 457
 13593 hanzi, 93
 13594 head, 272, 433
 13595 head percolation rules, 260
 13596 head rules, 271
 13597 head word, 228, 260, 271, 377
 13598 hedging, 443
 13599 held-out data, 155
 13600 Hessian matrix, 54
 13601 hidden Markov models, 170
 13602 hierarchical clustering, 352
 13603 hierarchical recurrent network, 493
 13604 hierarchical softmax, 357
 13605 hierarchical topic segmentation, 401
 13606 highway network, 68
 13607 holonymy, 91
 13608 homonymy, 89
 13609 human computation, 107
 13610 hypergraph, 412
 13611 hypernymy, 91, 295
 13612 hyperparameter, 38
 13613 hyponymy, 91
 13614 illocutionary force, 203
 13615 implicit discourse relations, 406
 13616 importance sampling, 474
 13617 importance score, 474
 13618 incremental perceptron, 290, 384
 13619 independent and identically distributed
 (IID), 32
 13621 indicator function, *see* delta function
 13622 indicator random variable, 502
 13623 inference, 163
 13624 inference (logic), 299
 13625 inference rules, 302
 13626 inflection
 affixes, 94
 13628 inflection point, 508
 13629 information extraction, 423
 13630 information retrieval, 17, 437
 13631 initiative (dialogue systems), 490
 13632 input word embeddings, 151
 13633 inside recurrence, 251, 252, 256
 13634 inside-outside algorithm, 255, 264
 13635 instance (AMR), 335
 13636 instance labels, 32
 13637 integer linear program, 448, 488
 13638 integer linear programming, 331, 382,
 415, 429
 13640 inter-annotator agreement, 106
 13641 interjections, 193
 13642 interlingua, 454
 13643 interpolation, 147, 215
 13644 interval algebra, 442
 13645 intrinsic evaluation, 359
 13646 inverse document frequency, 427
 13647 inverse relation (AMR), 336
 13648 inversion (of finite state automata), 219
 13649 irrealis, 86
 13650 Jensen's inequality, 115
 13651 joint probabilities, 503
 13652 Kalman smoother, 188
 13653 kernel function, 434

- 13654 kernel methods, 64
13655 kernel support vector machine, 64
13656 Kleene star, 208
13657 knapsack problem, 415
13658 knowledge base, 423
13659 knowledge base population, 436
- 13660 label bias, 294
13661 label bias problem, 289
13662 label propagation, 126, 136
13663 labeled dependencies, 273
13664 Lagrange multiplier, 509
13665 Lagrangian, 509
13666 Lagrangian dual, 509
13667 lambda calculus, 307
13668 lambda expressions, 307
13669 language model, 16, 142
13670 latent conditional random fields, 315
13671 latent Dirichlet allocation, 400
13672 latent semantic analysis, 348, 350
13673 latent variable, 114, 222, 314, 439, 454
13674 perceptron, 223
13675 latent variable perceptron, 315, 380
13676 layer normalization, 77, 469
13677 learning
13678 active, 134
13679 batch, 41
13680 constraint-driven, 134
13681 deep, 63
13682 discriminative, 40
13683 multiple instance, 134
13684 multitask, 134
13685 online, 41, 54
13686 semi-supervised, 92, 111
13687 transfer, 134
13688 unsupervised, 87, 111
13689 learning rate, 53
13690 learning to search, 271, 291, 386
13691 least squares, 88
13692 leave-one-out cross-validation, 39
13693 lemma (lexical semantics), 218
- 13694 lemma (morphology), 89
13695 lemmatization, 95
13696 lexical
13697 semantics, 89
13698 lexical entry, 308
13699 lexical unit (frame semantics), 326
13700 lexicalization
13701 in parsing, 260
13702 in text generation, 479
13703 lexicalized tree-adjoining grammar for
13704 discourse (D-LTAG), 405
13705 lexicon, 308
13706 in combinatory categorial grammar,
13707 237
13708 lexicon-based classification, 88
13709 seed, 89
13710 light verb, 337
13711 likelihood, 500
13712 linear separability, 41
13713 linearization, 488, 494
13714 link function, 57
13715 literal character, 208
13716 local minimum, 508
13717 local optimum, 118
13718 locally-normalized objective, 289
13719 log-bilinear language model, 363
13720 log-likelihood
13721 conditional, 68
13722 logistic function, 57
13723 long short-term memory (LSTM), 68,
13724 151, 153, 197, 464
13725 lookup layer, 69, 151
13726 loss
13727 function, 43
13728 hinge, 44
13729 logistic, 51
13730 zero-one, 44
13731 LSTM-CRF, 185, 334
- 13732 machine learning, 14
13733 supervised, 32

- 13734 theory, 42
- 13735 machine reading, 445
- 13736 machine translation, 141
- 13737 macro-reading, 424
- 13738 margin, 41, 46
 - functional, 46
 - geometric, 46
- 13741 marginal relevance, 487
- 13742 marginalize, 499
- 13743 markable, 378
- 13744 Markov assumption, 170
- 13745 Markov blanket, 170
- 13746 Markov Chain Monte Carlo (MCMC), 119, 131, 188
- 13748 Markov decision process, 490
- 13749 Markov random fields, 178
- 13750 matrix-tree theorem, 282
- 13751 max pooling, 81
- 13752 max-margin Markov network, 177
- 13753 max-product algorithm, 173
- 13754 maximum a posteriori, 38, 505
- 13755 maximum conditional likelihood, 50
- 13756 maximum entropy, 57
- 13757 maximum likelihood, 32, 36, 504
- 13758 maximum spanning tree, 279
- 13759 McNemar’s test, 100
- 13760 meaning representation, 299
- 13761 membership problem (for formal languages), 207
- 13763 memory cell (LSTM), 153
- 13764 mention
 - in coreference resolution, 371
 - in entity linking, 423
 - in information extraction, 425
- 13768 mention ranking, 380
- 13769 mention-pair model, 379
- 13770 meronymy, 91
- 13771 meteor, 457
- 13772 method of moments, 132
- 13773 micro-reading, 424
- 13774 mildly context-sensitive languages, 235
- 13775 minibatch, 54
- 13776 minimization (of finite-state automata), 212
- 13778 minimum error-rate training (MERT), 473
- 13779 minimum risk training, 473
- 13781 mixed-initiative, 490
- 13782 modality, 443
- 13783 model (machine learning), 19
- 13784 model builder, 305
- 13785 model checker, 305
- 13786 model-theoretic semantics, 300
- 13787 modeling (machine learning), 53
- 13788 modifier (dependency grammar), 272
- 13789 modus ponens, 302
- 13790 moment-matching, 57
- 13791 monomorphemic, 212
- 13792 morpheme, 17, 157, 211
- 13793 morphological segmentation, 175
- 13794 morphology, 95, 174, 210, 362, 470
 - derivational, 210
 - inflectional, 193, 210, 218
- 13797 morphosyntactic, 192
- 13798 attributes, 196
- 13799 morphotactic, 211
- 13800 multi-document summarization, 488
- 13801 multi-view learning, 124
- 13802 multiclass classification
 - one-versus-all, 434
 - one-versus-one, 434
- 13805 multilayer perceptron, 66
- 13806 multinomial
 - distribution, 34
 - Naïve Bayes, 34
- 13809 multiple instance learning, 439
- 13810 multiple-choice question answering, 445
- 13811 Naïve Bayes, 33
- 13812 name dictionary, 426
- 13813 named entities, 199
- 13814 named entity

- 13815 recognition, 185
 13816 named entity linking, 425
 13817 named entity recognition, 423, 425
 13818 named entity types, 425
 13819 nearest-neighbor, 64, 434
 13820 negation, 86, 443
 13821 negative sampling, 357, 358, 431
 13822 Neo-Davidsonian event semantics, 322
 13823 neural attention, 465
 13824 neural machine translation, 454
 13825 neural network, 64
 convolutional, 69, 78, 201
 feedforward, 66
 recurrent, 151
 13829 NIL entity, 425
 13830 noise-contrastive estimation, 151
 13831 noisy channel model, 142, 458
 13832 nominal modifier, 231
 13833 nominals, 371
 13834 nominals (coreference), 378
 13835 non-core roles (AMR), 336
 13836 non-terminals (context-free grammars),
 225
 13838 norm
 Frobenius, 73
 13840 normalization, text, 94
 13841 noun phrase, 14, 228
 13842 nouns, 192
 13843 NP-hard, 56, 429
 13844 nuclearity (RST), 409
 13845 nucleus (RST), 409
 13846 null hypothesis, 100
 13847 null subjects, 395
 13848 numeral (part of speech), 195
 13849 numerical optimization, 20
 13850 offset feature, 31
 13851 one-hot, 392
 13852 one-hot vector, 69
 13853 one-tailed p-value, 101
 13854 ontology, 20
 13855 open information extraction, 440
 13856 open word classes, 192
 13857 opinion polarity, 85
 13858 optimization
 batch, 53
 constrained, 47
 convex, 53
 quasi-Newton, 54
 13863 oracle, 288, 338
 13864 oracle (learning to search), 386
 13865 orthography, 212, 220
 13866 orthonormal matrix, 75
 13867 out-of-vocabulary words, 197
 13868 outside recurrence, 252, 256
 13869 overfitting, 38, 42
 13870 overgeneration, 219, 229
 13871 parallel corpora, 457
 13872 parameters, 504
 13873 paraphrase, 366
 13874 parent annotation, 259
 13875 parsing, 225
 13876 part-of-speech, 17, 191
 13877 part-of-speech tagging, 161
 13878 partially observable Markov decision
 process (POMDP), 492
 13879 partially supervised learning, 264
 13880 particle (part-of-speech), 195, 233
 13882 partition, 499
 13883 partition function, 180
 13884 passive-aggressive, 509
 13885 path (finite state automata), 209
 13886 Penn Discourse Treebank (PDTB), 405
 13887 Penn Treebank, 156, 175, 196, 228, 254
 13888 perceptron, 41
 13889 perplexity, 156
 13890 phonology, 212
 13891 phrase (syntax), 228
 13892 phrase-structure grammar, 228
 13893 pivot features, 128
 13894 pointwise mutual information, 294, 350

- 13895 policy, 385, 491
 13896 policy (search), 290
 13897 policy gradient, 386
 13898 polysemy, 90
 13899 pooling, 393
 13900 pooling (convolution), 79, 393, 482
 13901 positional encodings, 469
 13902 positive pointwise mutual information, 351
 13903 posterior, 500
 13905 power law, 14
 13906 pragmatics, 371
 13907 pre-trained word representations, 361
 13908 precision, 98, 500
 at-*k*, 99
 13910 labeled, 246
 13911 unlabeled, 246
 13912 precision-at-*k*, 418
 13913 precision-recall curve, 438
 13914 precision-recall curves, 99
 13915 predicate, 423
 13916 predicative adjectives, 233
 13917 predictive likelihood, 119
 13918 prepositional phrase, 14, 233
 13919 primal form, 509
 13920 principle of compositionality, 306
 13921 prior, 500
 13922 prior expectation, 505
 13923 probabilistic models, 504
 13924 probabilistic topic model, 429
 13925 probability
 conditional, 50, 499, 503
 joint, 32, 50
 marginal, 503
 13929 probability density function, 503
 13930 probability distribution, 502
 13931 probability mass function, 101, 503
 13932 probability simplex, 33
 13933 processes, 442
 13934 productivity, 211
 13935 projectivity, 275
 13936 pronominal anaphora resolution, 371
 13937 pronoun, 194
 13938 PropBank, 326
 13939 proper nouns, 193
 13940 property (logic), 303
 13941 proposal distribution, 474
 13942 proposition, 442
 13943 propositions, 300, 301
 13944 prosody, 203
 13945 proto-roles, 325
 13946 pseudo-projective dependency parsing, 285
 13947 pumping lemma, 223
 13949 pushdown automata, 225
 13950 pushdown automaton, 266
 13951 quadratic program, 47
 13952 quantifier, 303
 13953 quantifier, existential, 304
 13954 quantifier, universal, 304
 13955 question answering, 365, 425
 13956 random outcomes, 497
 13957 ranking, 426
 13958 ranking loss, 426
 13959 recall, 98, 500
 labeled, 246
 unlabeled, 246
 13960 recall-at-*k*, 419
 13963 receiver operating characteristic (ROC), 99
 13964 rectified linear unit (ReLU), 67
 13966 recurrent neural networks, 435
 13967 recursion, 14
 13968 recursive neural network, 415
 13969 recursive neural networks, 266, 363, 366
 13970 reference arguments, 341
 13971 reference resolution, 371
 13972 reference translations, 455
 13973 referent, 371
 13974 referring expression, 401

- 13975 referring expressions, 371, 480
 13976 reflexive pronoun, 373
 13977 regression, 88
 linear, 88
 13979 logistic, 50
 13980 ridge, 88
 13981 regular expression, 208
 13982 regular language, 208
 13983 regularization, 49
 13984 reification (events), 321
 13985 reinforcement learning, 385, 472
 13986 relation (logic), 309
 13987 relation extraction, 292, 339, 431
 13988 relations, 423
 13989 relations (information extraction), 423
 13990 relations (logic), 300
 13991 relative frequency estimate, 36, 142, 505
 13992 ReLU, *see* rectified linear unit
 leaky, 67
 13994 reranking, 265
 13995 residual networks, 68
 13996 retrofitting (word embeddings), 364
 13997 Rhetorical Structure Theory (RST), 408
 13998 rhetorical zones, 401
 13999 RIBES (translation metric), 457
 14000 risk, 473
 14001 roll-in (reinforcement learning), 386
 14002 roll-out (reinforcement learning), 387
 14003 root (morpheme), 363
 14004 saddle point, 74, 508
 14005 sample space, 497
 14006 satellite (RST), 409
 14007 satisfaction (logic), 305
 14008 scheduled sampling, 472
 14009 schema, 423, 424, 440
 14010 search error, 267, 384
 14011 second-order dependency parsing, 277
 14012 second-order logic, 304
 14013 segmented discourse representation
theory (SDRT), 404
 14015 self-attention, 468
 14016 self-training, 124
 14017 semantic, 192
 14018 semantic concordance, 92
 14019 semantic parsing, 306
 14020 semantic role, 322
 14021 Semantic role labeling, 322
 14022 semantic role labeling, 432, 440
 14023 semantic underspecification, 317
 14024 semantics, 258, 299
 14025 semi-supervised learning, 121, 360
 14026 semiring, 215
 algebra, 188
 expectation, 223
 tropical, 189
 14027 senses, 325
 14028 sentence (logic), 304
 14029 sentence compression, 487
 14030 sentence fusion, 488
 14031 sentence summarization, 486
 14032 sentiment, 85
 14033 sentiment analysis
 lexicon-based, 86
 targeted, 87
 14034 sentiment lexicon, 32
 14035 sequence-to-sequence, 464
 14036 shift-reduce parsing, 266
 14037 shifted positive pointwise mutual
information, 358
 14038 shortest-path algorithm, 213
 14039 sigmoid, 65
 14040 simplex, 131
 14041 singular value decomposition, 75, 120
 truncated, 120
 14042 singular vectors, 76
 14043 skipgram word embeddings, 356
 14044 slack variables, 48
 14045 slot filling, 436
 14046 slots (dialogue systems), 489
 14047 smooth functions, 45
 14048 smoothing, 38, 145

- 14056 Jeffreys-Perks, 145
 14057 Kneser-Ney, 149
 14058 Laplace, 38, 145
 14059 Lidstone, 145
 14060 softmax, 65, 150, 436
 hierarchical, 151
 14062 source language, 453
 14063 spanning tree, 272
 14064 sparse matrix, 350
 14065 sparsity, 56
 14066 spectral learning, 133
 14067 speech acts, 203
 14068 speech recognition, 141
 14069 split constituents, 328
 14070 spurious ambiguity, 283, 288, 312
 14071 squashing function, 151
 14072 squashing functions, 67
 14073 stand-off annotations, 106
 14074 Stanford Natural Language Inference
 corpus, 366
 14076 statistical machine translation, 454
 14077 statistical significance, 100
 14078 stem (morphology), 18, 211
 14079 stemmer, 94
 14080 step size, 508
 14081 stochastic gradient descent, 45, 54
 14082 stoplist, 96
 14083 stopwords, 96
 14084 string (formal language theory), 207
 14085 string-to-tree translation, 463
 14086 strong compositionality criterion (RST),
 411
 14088 structure induction, 186
 14089 structured attention, 484
 14090 structured perceptron, 176
 14091 structured prediction, 30
 14092 structured support vector machine, 177
 14093 subgradient, 45, 56
 14094 subjectivity detection, 87
 14095 subordinating conjunctions, 194
 14096 subordinating discourse relations, 409
 14097 sum-product algorithm, 180
 14098 summarization, 141, 413
 14099 supersenses, 360
 14100 support vector machine, 47
 kernel, 434
 14102 support vectors, 47
 14103 surface form, 219
 14104 surface realization, 479
 14105 synchronous context-free grammar, 463
 14106 synonymy, 90, 295, 345
 14107 synset (synonym set), 90
 14108 synsets, 390
 14109 syntactic dependencies, 272
 14110 syntactic path, 329
 14111 syntactic-semantic grammar, 307
 14112 syntax, 191, 227, 299
 14113 tagset, 192
 14114 tanh activation function, 67
 14115 target language, 453
 14116 tense, 193
 14117 terminal symbols (context-free
 grammars), 225
 14118 test set, 39, 111
 14120 test statistic, 100
 14121 text mining, 17
 14122 text planning, 479
 14123 thematic roles, 323
 14124 third axiom of probability, 498
 14125 third-order dependency parsing, 278
 TimeML, 441
 14127 tokenization, 93, 201
 14128 tokens and types, 34
 14129 topic models, 17
 14130 topic segmentation, 399
 14131 trace (syntax), 238
 14132 training set, 32, 111
 14133 transduction, 208
 14134 transformer architecture, 468
 14135 transition system, 283
 14136 transition-based parsing, 241, 266

- 14137 transitive closure, 381
 14138 translation error rate (TER), 457
 14139 translation model, 142
 14140 transliteration, 471
 14141 tree-adjoining grammar, 236
 14142 tree-to-string translation, 464
 14143 tree-to-tree translation, 463
 14144 treebank, 254
 14145 trellis, 166, 221
 14146 trigrams, 40
 14147 trilexical dependencies, 263
 14148 tropical semiring, 215
 14149 true negative, 98
 14150 true positive, 98, 501
 rate, 99
 14152 truncated singular value decomposition, 350
 14153 truth conditions, 305
 14155 tuning set, *see* development set, 97
 14156 Turing test, 15
 14157 two-tailed test, 102
 14158 type systems, 310
 14159 type-raising, 237, 310
 14160 unary closure, 245
 14161 undercut (argumentation), 413, 419
 14162 underfitting, 38
 14163 underflow, 33
 14164 undergeneration, 219, 229
 14165 Universal Dependencies, 192, 271
 14166 unseen word, 185
 14167 utterances, 203
 14168 validation function (semantic parsing), 316
 14169 validity (logic), 305
 14171 value function, 491
 14172 value iteration, 491
 14173 variable (AMR), 335
 14174 variable (logic), 303
 14175 variable, bound (logic), 304
 14176 variable, free (logic), 303
 14177 variance, 103
 in machine learning theory, 38
 14179 variational autoencoder, 487
 14180 Vauquois Pyramid, 454
 14181 verb phrase, 229
 14182 VerbNet, 324
 14183 verbs, 193
 14184 vertical Markovization, 259
 14185 Viterbi
 algorithm, 164
 variable, 166
 14186 volition (semantics), 323
 14189 WARP loss, 431
 14190 weight decay, 73
 14191 weighted context-free grammars, 244
 14192 Wikification, 425
 14193 Winograd schemas, 15
 14194 word
 senses, 89
 14196 word embedding, 69
 14197 word embeddings, 63, 152, 346, 347
 14198 word representations, 346
 14199 word sense disambiguation, 89
 14200 word tokens, 93
 14201 WordNet, 20
 14202 world model, 300
 14203 yield (context-free grammars), 225
 14204 Zipf's law, 159