

1

Natural Language Processing

2

Jacob Eisenstein

3

May 28, 2018

4 Contents

5	Contents	1
6	1 Introduction	13
7	1.1 Natural language processing and its neighbors	13
8	1.2 Some 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 Roadmap	23
15	I Words, bags of words, and features	25
16	2 Linear text classification	27
17	2.1 Naïve Bayes	30
18	2.1.1 Types and tokens	32
19	2.1.2 Prediction	33
20	2.1.3 Estimation	34
21	2.1.4 Smoothing and MAP estimation	36
22	2.1.5 Setting hyperparameters	36
23	2.2 Discriminative learning	37
24	2.2.1 Perceptron	38
25	2.2.2 Averaged perceptron	40
26	2.3 Loss functions and large-margin classification	41
27	2.3.1 Large margin classification	44
28	2.3.2 Support vector machines	45
29	2.3.3 Slack variables	46
30	2.4 Logistic regression	48
31	2.4.1 Regularization	49

32	2.4.2 Gradients	50
33	2.5 Optimization	50
34	2.5.1 Batch optimization	51
35	2.5.2 Online optimization	52
36	2.6 *Additional topics in classification	54
37	2.6.1 Feature selection by regularization	54
38	2.6.2 Other views of logistic regression	54
39	2.7 Summary of learning algorithms	56
40	3 Nonlinear classification	59
41	3.1 Feedforward neural networks	60
42	3.2 Designing neural networks	62
43	3.2.1 Activation functions	62
44	3.2.2 Network structure	63
45	3.2.3 Outputs and loss functions	64
46	3.2.4 Inputs and lookup layers	65
47	3.3 Learning neural networks	65
48	3.3.1 Backpropagation	67
49	3.3.2 Regularization and dropout	69
50	3.3.3 *Learning theory	70
51	3.3.4 Tricks	71
52	3.4 Convolutional neural networks	73
53	4 Linguistic applications of classification	79
54	4.1 Sentiment and opinion analysis	79
55	4.1.1 Related problems	81
56	4.1.2 Alternative approaches to sentiment analysis	82
57	4.2 Word sense disambiguation	83
58	4.2.1 How many word senses?	84
59	4.2.2 Word sense disambiguation as classification	85
60	4.3 Design decisions for text classification	86
61	4.3.1 What is a word?	86
62	4.3.2 How many words?	89
63	4.3.3 Count or binary?	90
64	4.4 Evaluating classifiers	90
65	4.4.1 Precision, recall, and <i>F</i> -MEASURE	91
66	4.4.2 Threshold-free metrics	93
67	4.4.3 Classifier comparison and statistical significance	94
68	4.4.4 *Multiple comparisons	97
69	4.5 Building datasets	97
70	4.5.1 Metadata as labels	98

71	4.5.2 Labeling data	98
72	5 Learning without supervision	105
73	5.1 Unsupervised learning	105
74	5.1.1 K -means clustering	106
75	5.1.2 Expectation Maximization (EM)	108
76	5.1.3 EM as an optimization algorithm	112
77	5.1.4 How many clusters?	113
78	5.2 Applications of expectation-maximization	114
79	5.2.1 Word sense induction	114
80	5.2.2 Semi-supervised learning	115
81	5.2.3 Multi-component modeling	116
82	5.3 Semi-supervised learning	117
83	5.3.1 Multi-view learning	118
84	5.3.2 Graph-based algorithms	119
85	5.4 Domain adaptation	120
86	5.4.1 Supervised domain adaptation	121
87	5.4.2 Unsupervised domain adaptation	122
88	5.5 *Other approaches to learning with latent variables	124
89	5.5.1 Sampling	124
90	5.5.2 Spectral learning	126
91	II Sequences and trees	133
92	6 Language models	135
93	6.1 N -gram language models	136
94	6.2 Smoothing and discounting	139
95	6.2.1 Smoothing	139
96	6.2.2 Discounting and backoff	140
97	6.2.3 *Interpolation	141
98	6.2.4 *Kneser-Ney smoothing	143
99	6.3 Recurrent neural network language models	144
100	6.3.1 Backpropagation through time	146
101	6.3.2 Hyperparameters	147
102	6.3.3 Gated recurrent neural networks	147
103	6.4 Evaluating language models	149
104	6.4.1 Held-out likelihood	149
105	6.4.2 Perplexity	150
106	6.5 Out-of-vocabulary words	151

107	7 Sequence labeling	153
108	7.1 Sequence labeling as classification	153
109	7.2 Sequence labeling as structure prediction	155
110	7.3 The Viterbi algorithm	157
111	7.3.1 Example	160
112	7.3.2 Higher-order features	161
113	7.4 Hidden Markov Models	161
114	7.4.1 Estimation	163
115	7.4.2 Inference	163
116	7.5 Discriminative sequence labeling with features	165
117	7.5.1 Structured perceptron	168
118	7.5.2 Structured support vector machines	168
119	7.5.3 Conditional random fields	170
120	7.6 Neural sequence labeling	175
121	7.6.1 Recurrent neural networks	175
122	7.6.2 Character-level models	177
123	7.6.3 Convolutional Neural Networks for Sequence Labeling	178
124	7.7 *Unsupervised sequence labeling	178
125	7.7.1 Linear dynamical systems	180
126	7.7.2 Alternative unsupervised learning methods	180
127	7.7.3 Semiring Notation and the Generalized Viterbi Algorithm	180
128	8 Applications of sequence labeling	183
129	8.1 Part-of-speech tagging	183
130	8.1.1 Parts-of-Speech	184
131	8.1.2 Accurate part-of-speech tagging	188
132	8.2 Morphosyntactic Attributes	190
133	8.3 Named Entity Recognition	191
134	8.4 Tokenization	193
135	8.5 Code switching	194
136	8.6 Dialogue acts	195
137	9 Formal language theory	197
138	9.1 Regular languages	198
139	9.1.1 Finite state acceptors	199
140	9.1.2 Morphology as a regular language	200
141	9.1.3 Weighted finite state acceptors	202
142	9.1.4 Finite state transducers	207
143	9.1.5 *Learning weighted finite state automata	212
144	9.2 Context-free languages	213
145	9.2.1 Context-free grammars	214

146	9.2.2 Natural language syntax as a context-free language	217
147	9.2.3 A phrase-structure grammar for English	219
148	9.2.4 Grammatical ambiguity and weighted context-free grammars	224
149	9.3 *Mildly context-sensitive languages	225
150	9.3.1 Context-sensitive phenomena in natural language	226
151	9.3.2 Combinatory categorial grammar	226
152	10 Context-free Parsing	231
153	10.1 Deterministic bottom-up parsing	232
154	10.1.1 Recovering the parse tree	234
155	10.1.2 Non-binary productions	234
156	10.1.3 Complexity	234
157	10.2 Ambiguity	235
158	10.2.1 Parser evaluation	236
159	10.2.2 Local solutions	237
160	10.3 Weighted Context-Free Grammars	238
161	10.3.1 Parsing with weighted context-free grammars	240
162	10.3.2 Probabilistic context-free grammars	241
163	10.4 Learning weighted context-free grammars	243
164	10.4.1 Probabilistic context-free grammars	243
165	10.4.2 Feature-based parsing	244
166	10.4.3 *Conditional random field parsing	245
167	10.4.4 Neural context-free grammars	246
168	10.5 Grammar refinement	247
169	10.5.1 Parent annotations and other tree transformations	247
170	10.5.2 Lexicalized context-free grammars	249
171	10.5.3 *Refinement grammars	253
172	10.6 Beyond context-free parsing	254
173	10.6.1 Reranking	255
174	10.6.2 Transition-based parsing	256
175	11 Dependency Parsing	259
176	11.1 Dependency grammar	259
177	11.1.1 Heads and dependents	260
178	11.1.2 Labeled dependencies	262
179	11.1.3 Dependency subtrees and constituents	262
180	11.2 Graph-based dependency parsing	264
181	11.2.1 Graph-based parsing algorithms	266
182	11.2.2 Computing scores for dependency arcs	267
183	11.2.3 Learning	269
184	11.3 Transition-based dependency parsing	270

185	11.3.1 Transition systems for dependency parsing	271
186	11.3.2 Scoring functions for transition-based parsers	275
187	11.3.3 Learning to parse	276
188	11.4 Applications	279
189	11.5 Additional reading	280
190	III Meaning	283
191	12 Logical semantics	285
192	12.1 Meaning and denotation	286
193	12.2 Logical representations of meaning	287
194	12.2.1 Propositional logic	287
195	12.2.2 First-order logic	288
196	12.3 Semantic parsing and the lambda calculus	292
197	12.3.1 The lambda calculus	292
198	12.3.2 Quantification	295
199	12.4 Learning semantic parsers	297
200	12.4.1 Learning from derivations	298
201	12.4.2 Learning from logical forms	300
202	12.4.3 Learning from denotations	302
203	13 Predicate-argument semantics	307
204	13.1 Semantic roles	309
205	13.1.1 VerbNet	310
206	13.1.2 Proto-roles and PropBank	311
207	13.1.3 FrameNet	312
208	13.2 Semantic role labeling	314
209	13.2.1 Semantic role labeling as classification	314
210	13.2.2 Semantic role labeling as constrained optimization	317
211	13.2.3 Neural semantic role labeling	320
212	13.3 Abstract Meaning Representation	323
213	13.3.1 AMR Parsing	325
214	13.4 Applications of Predicate-Argument Semantics	327
215	14 Distributional and distributed semantics	333
216	14.1 The distributional hypothesis	333
217	14.2 Design decisions for word representations	335
218	14.2.1 Representation	335
219	14.2.2 Context	336
220	14.2.3 Estimation	337

221	14.3 Latent semantic analysis	338
222	14.4 Brown clusters	339
223	14.5 Neural word embeddings	342
224	14.5.1 Continuous bag-of-words (CBOW)	343
225	14.5.2 Skipgrams	344
226	14.5.3 Computational complexity	344
227	14.5.4 Word embeddings as matrix factorization	346
228	14.6 Evaluating word embeddings	347
229	14.6.1 Intrinsic evaluations	348
230	14.6.2 Extrinsic evaluations	348
231	14.7 Distributed representations beyond distributional statistics	349
232	14.7.1 Word-internal structure	349
233	14.7.2 Lexical semantic resources	351
234	14.8 Distributed representations of multiword units	352
235	14.8.1 Purely distributional methods	352
236	14.8.2 Distributional-compositional hybrids	352
237	14.8.3 Supervised compositional methods	353
238	14.8.4 Hybrid distributed-symbolic representations	354
239	15 Reference Resolution	359
240	15.1 Forms of referring expressions	360
241	15.1.1 Pronouns	360
242	15.1.2 Proper Nouns	365
243	15.1.3 Nominals	366
244	15.2 Algorithms for coreference resolution	366
245	15.2.1 Mention-pair models	367
246	15.2.2 Mention-ranking models	368
247	15.2.3 Transitive closure in mention-based models	369
248	15.2.4 Entity-based models	370
249	15.3 Representations for coreference resolution	375
250	15.3.1 Features	376
251	15.3.2 Distributed representations of mentions and entities	378
252	15.4 Additional reading	381
253	16 Discourse	383
254	16.1 Segments	383
255	16.1.1 Topic segmentation	384
256	16.1.2 Functional segmentation	385
257	16.2 Entities and reference	385
258	16.2.1 Centering theory	386
259	16.2.2 The entity grid	387

260	16.2.3 *Formal semantics beyond the sentence level	388
261	16.3 Relations	388
262	16.3.1 Shallow discourse relations	389
263	16.3.2 Hierarchical discourse relations	392
264	16.3.3 Argumentation	396
265	16.3.4 Applications of discourse relations	397
266	IV Applications	403
267	17 Information extraction	405
268	17.1 Entities	407
269	17.1.1 Entity linking by learning to rank	408
270	17.1.2 Collective entity linking	410
271	17.1.3 *Pairwise ranking loss functions	411
272	17.2 Relations	413
273	17.2.1 Pattern-based relation extraction	414
274	17.2.2 Relation extraction as a classification task	414
275	17.2.3 Knowledge base population	418
276	17.2.4 Open information extraction	421
277	17.3 Events	422
278	17.4 Hedges, denials, and hypotheticals	424
279	17.5 Question answering and machine reading	426
280	17.5.1 Formal semantics	426
281	17.5.2 Machine reading	427
282	18 Machine translation	433
283	18.1 Machine translation as a task	433
284	18.1.1 Evaluating translations	435
285	18.1.2 Data	437
286	18.2 Statistical machine translation	438
287	18.2.1 Statistical translation modeling	439
288	18.2.2 Estimation	441
289	18.2.3 Phrase-based translation	442
290	18.2.4 *Syntax-based translation	443
291	18.3 Neural machine translation	444
292	18.3.1 Neural attention	446
293	18.3.2 *Neural machine translation without recurrence	448
294	18.3.3 Out-of-vocabulary words	450
295	18.4 Decoding	451
296	18.5 Training towards the evaluation metric	453

297	19 Text generation	457
298	19.1 Data-to-text generation	457
299	19.1.1 Latent data-to-text alignment	459
300	19.1.2 Neural data-to-text generation	460
301	19.2 Text-to-text generation	464
302	19.2.1 Neural abstractive summarization	464
303	19.2.2 Sentence fusion for multi-document summarization	466
304	19.3 Dialogue	467
305	19.3.1 Finite-state and agenda-based dialogue systems	467
306	19.3.2 Markov decision processes	468
307	19.3.3 Neural chatbots	470
308	A Probability	473
309	A.1 Probabilities of event combinations	473
310	A.1.1 Probabilities of disjoint events	474
311	A.1.2 Law of total probability	475
312	A.2 Conditional probability and Bayes' rule	475
313	A.3 Independence	477
314	A.4 Random variables	478
315	A.5 Expectations	479
316	A.6 Modeling and estimation	480
317	B Continuous optimization	483
318	B.1 Gradient descent	484
319	B.2 Constrained optimization	484
320	B.3 Example: passive-aggressive online learning	485
321	Bibliography	487

322 Notation

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

Basics

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

Linear algebra

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

Text datasets

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

Probabilities

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

Machine learning

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

327 **Chapter 1**

328 **Introduction**

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

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

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

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

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

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

359 **Machine Learning** Contemporary approaches to natural language processing rely heavily
360 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.

- Unlike images or audio, text data is fundamentally discrete, with meaning created by combinatorial arrangements of symbolic units. This is particularly consequential for applications in which text is the output, such as translation and summarization, because it is not possible to gradually approach an optimal solution.
 - Although the set of words is discrete, new words are always being created. Furthermore, the distribution over words (and other linguistic elements) resembles that of a **power law** (Zipf, 1949): there will be a few words that are very frequent, and a long tail of words that are rare. A consequence is that natural language processing algorithms must be especially robust to observations that do not occur in the training data.
 - Language is **recursive**: units such as words can combine to create phrases, which can combine by the very same principles to create larger phrases. For example, a **noun phrase** can be created by combining a smaller noun phrase with a **prepositional phrase**, as in *the horrid aspect and revenge of the whale*. The prepositional phrase is created by combining a preposition (in this case, *of*) with another noun phrase (*the whale*). In this way, it is possible to create arbitrarily long phrases, such as,

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

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

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

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

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

- 407 (1.2) The trophy doesn’t fit into the brown suitcase because **it** is too [small/large].
- 408 When the final word is *small*, then the pronoun *it* refers to the suitcase; when the final
 409 word is *large*, then *it* refers to the trophy. Solving this example requires spatial reasoning;
 410 other schemas require reasoning about actions and their effects, emotions and intentions,
 411 and social conventions.
- 412 The Winograd schemas demonstrate that natural language understanding cannot be
 413 achieved in isolation from knowledge and reasoning. Yet the history of artificial intelli-
 414 gence has been one of increasing specialization: with the growing volume of research in
 415 subdisciplines such as natural language processing, machine learning, and computer vi-
 416 sion, it is difficult for anyone to maintain expertise across the entire field. Still, recent work

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

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

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

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

442 **Speech Processing** Natural language is often communicated in spoken form, and speech
443 recognition is the task of converting an audio signal to text. From one perspective, this
444 is a signal processing problem, which might be viewed as a preprocessing step before
445 natural language processing can be applied. However, contextual factors play a critical
446 role in speech recognition by human listeners: knowledge of the surrounding words in-
447 fluences perception and helps to correct for noise (Miller et al., 1951). For this reason,
448 speech recognition is often integrated with text analysis, particularly with statistical **lan-**
449 **guage model**, which quantify the probability of a sequence of text (see chapter 6). Beyond
450 speech recognition, the broader field of speech processing includes the study of speech-
451 based dialogue systems, which are briefly discussed in chapter 19. Historically, speech
452 processing has often been pursued in electrical engineering departments, while natural
453 language processing has been the purview of computer scientists. For this reason, the
454 extent of interaction between these two disciplines is less than it might otherwise be.

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

466 1.2 Some 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 op-
471 position between two extreme viewpoints on how process natural language, and in each
472 case, existing approaches can be placed on a continuum between these two extremes.

473 1.2.1 Learning and knowledge

474 A recurring debate in natural language processing is the relative importance of machine
475 learning and linguistic knowledge. On one extreme, advocates of “natural language pro-
476 cessing from scratch” (Collobert et al., 2011) propose to use machine learning to train
477 end-to-end systems that transmute raw text into any desired output structure: e.g., a sum-
478 mary, database, or translation. On the other extreme, the core work of natural language
479 processing is to transform language into a stack of general-purpose linguistic structures:
480 from subword units called **morphemes**, to word-level **parts-of-speech**, to tree-structured
481 representations of grammar, and beyond, to logic-based representations of meaning. In
482 theory, these general-purpose structures should then be able to support any desired lan-
483 guage technology application.

484 The end-to-end learning approach has been buoyed by recent results in computer vi-
485 sion and speech recognition, in which advances in machine learning have swept away
486 expert-engineered representations based on the fundamentals of optics and phonology (Krizhevsky
487 et al., 2012; Graves and Jaitly, 2014). But while some amount of machine learning is an el-
488 ement of nearly every contemporary approach to natural language processing, linguistic
489 representations such as syntax trees have not yet gone the way of the visual edge detector
490 or the auditory triphone. Linguists have argued for the existence of a “language faculty”

491 in all human beings, which encodes a set of abstractions specially designed to facilitate
 492 the understanding and production of languages. The argument for the existence of such
 493 a language faculty is based on the observation that children learn language faster and
 494 from fewer examples than would be reasonably possible, if language was learned from
 495 experience alone.³ Regardless of the cognitive validity of these arguments, it seems to
 496 be the case that modeling linguistic structure is particularly important in scenarios where
 497 training data is limited.

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

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

516 The debate about the relative importance of machine learning and linguistic knowl-
 517 edge sometimes becomes heated. No machine learning specialist likes to be told that their
 518 engineering methodology is unscientific alchemy;⁴ nor does a linguist want to hear that
 519 the search for general linguistic principles and structures has been made irrelevant by big
 520 data. Yet there is clearly room for both types of research: we need to know how far we
 521 can go with end-to-end learning alone, while at the same time, we continue the search for
 522 linguistic representations that generalize across applications, scenarios, and languages.
 523 For more on the history of this debate, see Church (2011); for an optimistic view of the
 524 potential symbiosis between computational linguistics and deep learning, see Manning
 525 (2015).

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

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

526 **1.2.2 Search and learning**

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

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

529 where,

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

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

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

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

552 **Learning.** The learning module is responsible for finding the parameters $\boldsymbol{\theta}$. This is typ-
 553 ically (but not always) done by processing a large dataset of labeled examples,
 554 $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$. Like search, learning is often approached through the framework
 555 of optimization, as we will see in chapter 2. Because the parameters are usually

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

continuous, learning algorithms generally rely on **numerical optimization**, searching over vectors of real numbers for parameters that optimize some function of the model and the labeled data. Some basic principles of numerical optimization are reviewed in Appendix B.

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

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

1.2.3 Relational, compositional, and distributional perspectives

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

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

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

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

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 analyzing entire texts
613 and dialogues through a single unified analytic lens, grounding out in the smallest parts
614 of individual words.

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

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

should also be familiar with probability and statistics. A review of basic probability is found in Appendix A, and a minimal review of numerical optimization is found in Appendix B. For linear algebra, the online course and textbook from Strang (2016) are an excellent source of review material. Deisenroth et al. (2018) are currently preparing a textbook on *Mathematics for Machine Learning*, and several chapters can be found online.⁶ For an introduction to probabilistic modeling and estimation, see James et al. (2013); for a more advanced and comprehensive discussion of the same material, the classic reference is Hastie et al. (2009).

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

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

1.3.2 Roadmap

[todo: add]

Acknowledgments

Several of my colleagues and students read early drafts of chapters in their areas of expertise, including Yoav Artzi, Kevin Duh, Heng Ji, Jessy Li, Brendan O'Connor, Yuval Pinter, Nathan Schneider, Pamela Shapiro, Noah A. Smith, Sandeep Soni, and Luke Zettlemoyer. I would also like to thank the following people for helpful discussions of the material: Kevin Murphy, Shawn Ling Ramirez, William Yang Wang, and Bonnie Webber. Several students and colleagues found mistakes in early drafts: Parminder Bhatia, Kimberly Caras, Chris Gu, Joshua Killingsworth, Jonathan May, Taha Merghani, Gus Monod, Raghavendra Murali, Nidish Nair, Brendan O'Connor, Yuval Pinter, Nathan Schneider,

⁶<https://mml-book.github.io/>

695 Zhewei Sun, Ashwin Cunnappakkam Vinjimir, Clay Washington, Ishan Waykul, and Yuyu
696 Zhang. Special thanks to the many students in Georgia Tech’s CS 4650 and 7650 who suf-
697 fered through early versions of the text.

698

Part I

699

Words, bags of words, and features

700

Chapter 2

701

Linear text classification

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

707 To perform this task, the first question is how to represent each document. A common
708 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
709 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
710 in the vocabulary.

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

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

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

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

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

728 As the notation suggests, f is a function of two arguments, the word counts \mathbf{x} and the
 729 label y , and it returns a vector output. For example, given arguments \mathbf{x} and y , element j
 730 of this feature vector might be,

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

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

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

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

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

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

734 where $\underbrace{[0; 0; \dots; 0]}_{(K-1) \times V}$ is a column vector of $(K - 1) \times V$ zeros, and the semicolon indicates
 735 vertical concatenation. This arrangement is shown in Figure 2.1; the notation may seem
 736 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 $\Psi(\mathbf{x}, y)$. This
 inner product gives a scalar measure of the compatibility of the observation \mathbf{x} with label
 y .¹ For any document \mathbf{x} , we predict the label \hat{y} ,

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

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

737 This inner product notation gives a clean separation between the *data* (\mathbf{x} and y) and the
 738 *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 $\Psi(\mathbf{x}, y = K) = 0$ regardless of \mathbf{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 $y = \text{Sign}(\beta \cdot \mathbf{x} + a)$, 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 $\theta \cdot f(\mathbf{x}, y)$ for all $y \in \mathcal{Y}$.

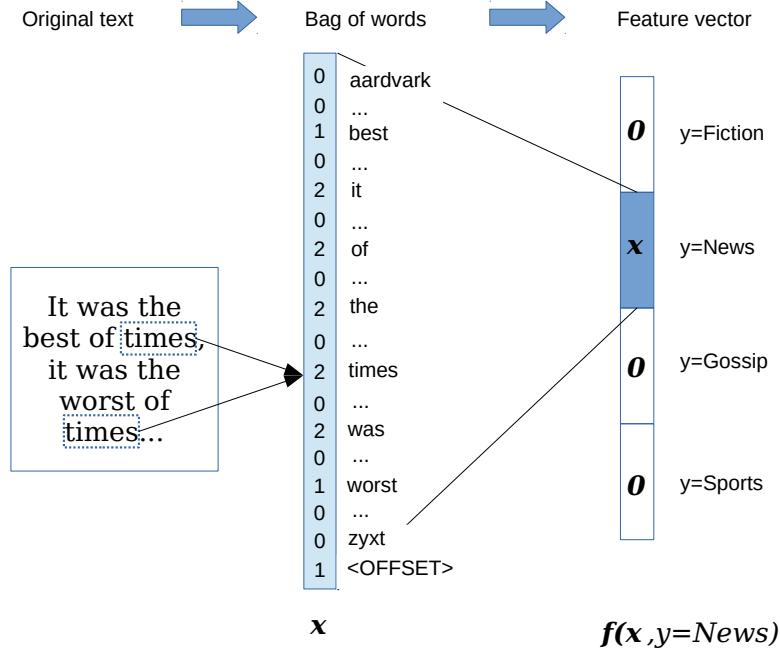


Figure 2.1: The bag-of-words and feature vector representations, for a hypothetical text classification task.

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

741 It is common to add an **offset feature** at the end of the vector of word counts x , which
 742 is always 1. We then have to also add an extra zero to each of the zero vectors, to make the
 743 vector lengths match. This gives the entire feature vector $f(x, y)$ a length of $(V + 1) \times K$.
 744 The weight associated with this offset feature can be thought of as a bias for or against
 745 each label. For example, if we expect most documents to be spam, then the weight for
 746 the offset feature for $y = \text{SPAM}$ should be larger than the weight for the offset feature for
 747 $y = \text{HAM}$.

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

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

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

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

758 2.1 Naïve Bayes

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

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

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

- 772 • The first line of this generative model encodes the assumption that the instances are
 773 mutually independent: neither the label nor the text of document i affects the label
 774 or text of document j .⁶ Furthermore, the instances are identically distributed: the
 775 distributions over the label $y^{(i)}$ and the text $\mathbf{x}^{(i)}$ (conditioned on $y^{(i)}$) are the same
 776 for all instances i .
- 777 • The second line of the generative model states that the random variable $y^{(i)}$ is drawn
 778 from a categorical distribution with parameter $\boldsymbol{\mu}$. Categorical distributions are like
 779 weighted dice: the vector $\boldsymbol{\mu} = [\mu_1, \mu_2, \dots, \mu_K]^\top$ gives the probabilities of each la-
 780 bel, so that the probability of drawing label y is equal to μ_y . For example, if $\mathcal{Y} =$
 781 $\{\text{POSITIVE}, \text{NEGATIVE}, \text{NEUTRAL}\}$, we might have $\boldsymbol{\mu} = [0.1, 0.7, 0.2]^\top$. We require
 782 $\sum_{y \in \mathcal{Y}} \mu_y = 1$ and $\mu_y \geq 0, \forall y \in \mathcal{Y}$.⁷
- 783 • The third line describes how the bag-of-words counts $\mathbf{x}^{(i)}$ are generated. By writing
 784 $\mathbf{x}^{(i)} | y^{(i)}$, 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 .

785 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:

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

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

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

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

798 2.1.1 Types and tokens

799 A slight modification to the generative model of Naïve Bayes is shown in Algorithm 2.
 800 Instead of generating a vector of counts of **types**, x , this model generates a *sequence of*
 801 **tokens**, $w = (w_1, w_2, \dots, w_M)$. The distinction between types and tokens is critical: $x_j \in$
 802 $\{0, 1, 2, \dots, M\}$ is the count of word type j in the vocabulary, e.g., the number of times
 803 the word *cannibal* appears; $w_m \in \mathcal{V}$ is the identity of token m in the document, e.g. $w_m =$
 804 *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  $w_m^{(i)} | y^{(i)} \sim \text{Categorical}(\boldsymbol{\phi}_{y^{(i)}})$ .

```

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

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

823 **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.

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

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

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

where

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

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

824 The feature function $\mathbf{f}(\mathbf{x}, y)$ is a vector of V word counts and an offset, padded by
825 zeros for the labels not equal to y (see Equations 2.3-2.5, and Figure 2.1). This construction
826 ensures that the inner product $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y)$ only activates the features whose weights are
827 in $\boldsymbol{\theta}^{(y)}$. These features and weights are all we need to compute the joint log-probability
828 $\log p(\mathbf{x}, y)$ for each y . This is a key point: through this notation, we have converted the
829 problem of computing the log-likelihood for a document-label pair (\mathbf{x}, y) into the compu-
830 tation of a vector inner product.

831 2.1.3 Estimation

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

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

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

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

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

(c) Jacob Eisenstein 2018. Work in progress.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

846 2.1.4 Smoothing and MAP estimation

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

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

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

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

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

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

867 2.1.5 Setting hyperparameters

868 How should we choose the best value of hyperparameters like α ? Maximum likelihood
 869 will not work: the maximum likelihood estimate of α on the training set will always be
 870 $\alpha = 0$. In many cases, what we really want is **accuracy**: the number of correct predictions,
 871 divided by the total number of predictions. (Other measures of classification performance
 872 are discussed in § 4.4.) As we will see, it is hard to optimize for accuracy directly. But for
 873 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.

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

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

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

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

900 2.2 Discriminative learning

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

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

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

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

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

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

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

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

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

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

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

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

920 The origin of the Naïve Bayes independence assumption is the learning objective,
 921 $p(\mathbf{x}^{(1:N)}, y^{(1:N)})$, which requires modeling the probability of the observed text. In clas-
 922 sification problems, we are always given \mathbf{x} , and are only interested in predicting the label
 923 y , so it seems unnecessary to model the probability of \mathbf{x} . **Discriminative learning** algo-
 924 rithms focus on the problem of predicting y , and do not attempt to model the probability
 925 of the text \mathbf{x} .

926 2.2.1 Perceptron

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

```

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

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

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

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

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

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

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

950 How useful is this proof? Minsky and Papert (1969) famously proved that the simple
 951 logical function of *exclusive-or* is not separable, and that a perceptron is therefore inca-
 952 pable of learning this function. But this is not just an issue for the perceptron: any linear
 953 classification algorithm, including Naïve Bayes, will fail on this task. In natural language
 954 classification problems usually involve high dimensional feature spaces, with thousands
 955 or millions of features. For these problems, it is very likely that the training data is indeed
 956 separable. And even if the data is not separable, it is still possible to place an upper bound
 957 on the number of errors that the perceptron algorithm will make (Freund and Schapire,
 958 1999).

959 2.2.2 Averaged perceptron

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

967 This procedure is shown in Algorithm 4. The learning algorithm is nearly identical,
 968 but we also maintain a vector of the sum of the weights, \mathbf{m} . At the end of the learning
 969 procedure, we divide this sum by the total number of updates t , to compute the average
 970 weights, $\bar{\theta}$. These average weights are then used for prediction. In the algorithm sketch,
 971 the average is computed from a running sum, $\mathbf{m} \leftarrow \mathbf{m} + \theta$. However, this is inefficient,
 972 because it requires $|\theta|$ operations to update the running sum. When $f(\mathbf{x}, y)$ is sparse,
 973 $|\theta| \gg |f(\mathbf{x}, y)|$ for any individual (\mathbf{x}, y) . This means that computing the running sum will
 974 be much more expensive than computing of the update to θ itself, which requires only
 975 $2 \times |f(\mathbf{x}, y)|$ operations. One of the exercises is to sketch a more efficient algorithm for
 976 computing the averaged weights.

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

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

Algorithm 4 Averaged perceptron learning algorithm

```

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

```

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

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

988 **2.3 Loss functions and large-margin classification**

989 Naïve Bayes chooses the weights $\boldsymbol{\theta}$ by maximizing the joint log-likelihood $\log p(\mathbf{x}^{(1:N)}, \mathbf{y}^{(1:N)})$.
 990 By convention, optimization problems are generally formulated as minimization of a **loss**
 991 **function**. The input to a loss function is the vector of weights $\boldsymbol{\theta}$, and the output is a non-
 992 negative scalar, measuring the performance of the classifier on a training instance. The
 993 loss $\ell(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)})$ is then a measure of the performance of the weights $\boldsymbol{\theta}$ on the instance
 994 $(\mathbf{x}^{(i)}, y^{(i)})$. The goal of learning is to minimize the sum of the losses across all instances in
 995 the training set.

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

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

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

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

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

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

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

998 Loss functions provide a general framework for comparing machine learning objec-
 999 tives. For example, an alternative loss function is the **zero-one loss**,

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

1000 The zero-one loss is zero if the instance is correctly classified, and one otherwise. The
 1001 sum of zero-one losses is proportional to the error rate of the classifier on the training
 1002 data. Since a low error rate is often the ultimate goal of classification, this may seem
 1003 ideal. But the zero-one loss has several problems. One is that it is **non-convex**,¹³ which
 1004 means that there is no guarantee that gradient-based optimization will be effective. A
 1005 more serious problem is that the derivatives are useless: the partial derivative with respect
 1006 to any parameter is zero everywhere, except at the points where $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y) = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, \hat{y})$
 1007 for some \hat{y} . At those points, the loss is discontinuous, and the derivative is undefined.

1008 The perceptron optimizes the following loss function:

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

1009 When $\hat{y} = y^{(i)}$, the loss is zero; otherwise, it increases linearly with the gap between the
 1010 score for the predicted label \hat{y} and the score for the true label $y^{(i)}$. Plotting this loss against
 1011 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
 1012 **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.

1013 To see why this is the loss function optimized by the perceptron, take the derivative
 1014 with respect to θ ,

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

1015 At each instance perceptron algorithm takes a step of magnitude one in the opposite direction
 1016 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,
 1017 this is an example of the optimization algorithm **stochastic gradient descent**, applied to
 1018 the objective in Equation 2.41.

1019 **Breaking ties with subgradient descent** Careful readers will notice the tacit assumption
 1020 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
 1021 that maximize this function? Consider binary classification: if the maximizer is $y^{(i)}$, then
 1022 the gradient is zero, and so is the perceptron update; if the maximizer is $\hat{y} \neq y^{(i)}$, then the
 1023 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
 1024 loss is not **smooth**, because the first derivative has a discontinuity at the hinge point,
 1025 where the score for the true label $y^{(i)}$ is equal to the score for some other label \hat{y} . At this
 1026 point, there is no unique gradient; rather, there is a set of **subgradients**. A vector v is a
 1027 subgradient of the function g at u_0 iff $g(u) - g(u_0) \geq v \cdot (u - u_0)$ for all u . Graphically,
 1028 this defines the set of hyperplanes that include $g(u_0)$ and do not intersect g at any other
 1029 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
 1030 approach from the right, the gradient is 0. At the hinge point, the subgradients include all
 1031 vectors that are bounded by these two extremes. In subgradient descent, *any* subgradient
 1032 can be used (Bertsekas, 2012). Since both 0 and $\mathbf{f}(\mathbf{x}, \hat{y}) - \mathbf{f}(\mathbf{x}, y)$ are subgradients at the
 1033 hinge point, either one can be used in the perceptron update.

1034 **Perceptron versus Naïve Bayes** The perceptron loss function has some pros and cons
 1035 with respect to the negative log-likelihood loss implied by Naïve Bayes.

- 1036 • Both ℓ_{NB} and $\ell_{\text{PERCEPTRON}}$ are convex, making them relatively easy to optimize. How-
 1037 ever, ℓ_{NB} can be optimized in closed form, while $\ell_{\text{PERCEPTRON}}$ requires iterating over
 1038 the dataset multiple times.
- 1039 • ℓ_{NB} can suffer **infinite** loss on a single example, since the logarithm of zero probabili-
 1040 ty is negative infinity. Naïve Bayes will therefore overemphasize some examples,
 1041 and underemphasize others.
- 1042 • $\ell_{\text{PERCEPTRON}}$ treats all correct answers equally. Even if θ only gives the correct answer
 1043 by a tiny margin, the loss is still zero.

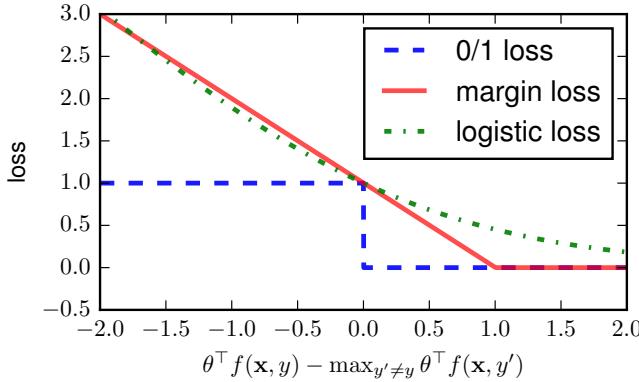


Figure 2.2: Margin, zero-one, and logistic loss functions.

1044 **2.3.1 Large margin classification**

1045 This last comment suggests a potential problem with the perceptron. Suppose a test ex-
 1046 ample is very close to a training example, but not identical. If the classifier only gets the
 1047 correct answer on the training example by a small margin, then it may get the test instance
 1048 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,

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

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

1049 where $(x)_+ = \max(0, x)$. The loss is zero if there is a margin of at least 1 between the
 1050 score for the true label and the best-scoring alternative \hat{y} . This is almost identical to the
 1051 perceptron loss, but the hinge point is shifted to the right, as shown in Figure 2.2. The
 1052 margin loss is a convex upper bound on the zero-one loss.

1053 **2.3.2 Support vector machines**

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

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

1054 This is a **constrained optimization** problem, where the second line describes constraints
 1055 on the space of possible solutions θ . In this case, the constraint is that the functional
 1056 margin always be at least one, and the objective is that the minimum geometric margin
 1057 be as large as possible.

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

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

1058 This optimization problem is a **quadratic program**: the objective is a quadratic function
 1059 of the parameters, and the constraints are all linear inequalities. The resulting classifier
 1060 is better known as the **support vector machine**. The name derives from one of the
 1061 solutions, which is to incorporate the constraints through Lagrange multipliers $\alpha_i \geq 0, i =$
 1062 $1, 2, \dots, N$. The instances for which $\alpha_i > 0$ are the **support vectors**; other instances are
 1063 irrelevant to the classification boundary.

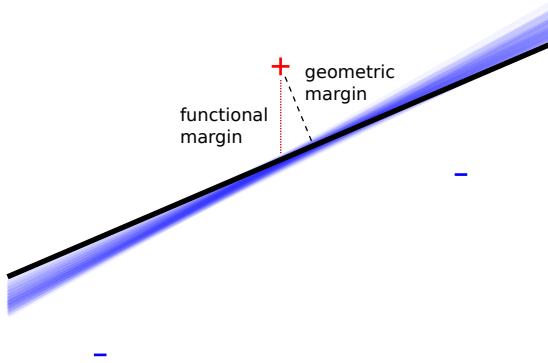


Figure 2.3: Functional and geometric margins for a binary classification problem. All separators that satisfy the margin constraint are shown. The separator with the largest geometric margin is shown in bold.

1064 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:

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

1065 The hyperparameter C controls the tradeoff between violations of the margin con-
 1066 straint and the preference for a low norm of θ . As $C \rightarrow \infty$, slack is infinitely expensive,
 1067 and there is only a solution if the data is separable. As $C \rightarrow 0$, slack becomes free, and
 1068 there is a trivial solution at $\theta = 0$. Thus, C plays a similar role to the smoothing parame-
 1069 ter in Naïve Bayes (§ 2.1.4), trading off between a close fit to the training data and better
 1070 generalization. Like the smoothing parameter of Naïve Bayes, C must be set by the user,
 1071 typically by maximizing performance on a heldout development set.

1072 To solve the constrained optimization problem defined in Equation 2.48, we can first

1073 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,

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

1074 where each ξ_i has been substituted by the right-hand side of Equation 2.49, and the factor
 1075 of C on the slack variables has been replaced by an equivalent factor of $\lambda = \frac{1}{C}$ on the
 1076 norm of the weights.

1077 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,

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

1078 This objective maximizes over all $y \in \mathcal{Y}$, in search of labels that are both *strong*, as measured by $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y)$, and *wrong*, as measured by $c(y^{(i)}, y)$. This maximization is known
 1079 as **cost-augmented inference**, because it augments the maximization objective to favor
 1080 high-cost predictions. If the highest-scoring label is $y = y^{(i)}$, then the margin constraint is
 1081 satisfied, and the loss for this instance is zero. Cost-augmentation is only for learning: it
 1082 is not applied when making predictions on unseen data.

Differentiating Equation 2.52 with respect to the weights gives,

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

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

1084 where L_{SVM} refers to minimization objective in Equation 2.52. This gradient is very similar
 1085 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 (Tsochantaridis et al., 2004). This idea is revisited in chapter 7.

1086 weights towards 0. The other difference is the cost $c(y^{(i)}, y)$, which is added to $\theta \cdot \mathbf{f}(\mathbf{x}, y)$
 1087 when choosing \hat{y} during training. This term derives from the margin constraint: large
 1088 margin classifiers learn not only from instances that are incorrectly classified, but also
 1089 from instances for which the correct classification decision was not sufficiently confident.

1090 2.4 Logistic regression

1091 Thus far, we have seen two broad classes of learning algorithms. Naïve Bayes is a prob-
 1092 abilistic method, where learning is equivalent to estimating a joint probability distribu-
 1093 tion. The perceptron and support vector machine are discriminative, error-driven algo-
 1094 rithms: the learning objective is closely related to the number of errors on the training
 1095 data. Probabilistic and error-driven approaches each have advantages: probability makes
 1096 it possible to quantify uncertainty about the predicted labels, but the probability model of
 1097 Naïve Bayes makes unrealistic independence assumptions that limit the features that can
 1098 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**,

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

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

1099 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.

1100 the sum, we have the (additive inverse of the) **logistic loss**,

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

1101 The logistic loss is shown in Figure 2.2. A key difference from the zero-one and hinge
 1102 losses is that logistic loss is never zero. This means that the objective function can always
 1103 be improved by assigning higher confidence to the correct label.

1104 2.4.1 Regularization

1105 As with the support vector machine, better generalization can be obtained by penalizing
 1106 the norm of $\boldsymbol{\theta}$. This is done by adding a term of $\frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2$ to the minimization objective.
 1107 This is called L_2 regularization, because $\|\boldsymbol{\theta}\|_2^2$ is the squared L_2 norm of the vector $\boldsymbol{\theta}$.
 1108 Regularization forces the estimator to trade off performance on the training data against
 1109 the norm of the weights, and this can help to prevent overfitting. Consider what would
 1110 happen to the unregularized weight for a base feature j that is active in only one instance
 1111 $\mathbf{x}^{(i)}$: the conditional log-likelihood could always be improved by increasing the weight
 1112 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
 1113 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,

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

1114 so that the regularization weight λ is equal to the inverse variance of the prior, $\lambda = \frac{1}{\sigma^2}$.

1115 **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,

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

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

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

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

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

1116 The final step employs the definition of a conditional expectation (§ A.5). The gradient of
 1117 the logistic loss is equal to the difference between the expected counts under the current
 1118 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
 1119 vectors are equal for a single instance, there is nothing more to learn from it; when they
 1120 are equal in sum over the entire dataset, there is nothing more to learn from the dataset as
 1121 a whole. The gradient of the hinge loss is nearly identical, but it involves the features of
 1122 the predicted label under the current model, $\mathbf{f}(\mathbf{x}^{(i)}, \hat{y})$, rather than the expected features
 1123 $E_{Y|X}[\mathbf{f}(\mathbf{x}^{(i)}, y)]$ under the conditional distribution $p(y | \mathbf{x}; \boldsymbol{\theta})$.

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

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

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

1124 **2.5 Optimization**

1125 Each of the classification algorithms in this chapter can be viewed as an optimization
 1126 problem:

- 1127 • In Naïve Bayes, the objective is the joint likelihood $\log p(\mathbf{x}^{(1:N)}, \mathbf{y}^{(1:N)})$. Maximum
 1128 likelihood estimation yields a closed-form solution for $\boldsymbol{\theta}$.

- 1129 • In the support vector machine, the objective is the regularized margin loss,

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

1130 There is no closed-form solution, but the objective is convex. The perceptron algo-
1131 rithm minimizes a similar objective.

- 1132 • In logistic regression, the objective is the regularized negative log-likelihood,

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

1133 Again, there is no closed-form solution, but the objective is convex.

1134 These learning algorithms are distinguished by *what* is being optimized, rather than
1135 *how* the optimal weights are found. This decomposition is an essential feature of con-
1136 temporary machine learning. The domain expert's job is to design an objective function
1137 — or more generally, a **model** of the problem. If the model has certain characteristics,
1138 then generic optimization algorithms can be used to find the solution. In particular, if an
1139 objective function is differentiable, then gradient-based optimization can be employed;
1140 if it is also convex, then gradient-based optimization is guaranteed to find the globally
1141 optimal solution. The support vector machine and logistic regression have both of these
1142 properties, and so are amenable to generic **convex optimization** techniques (Boyd and
1143 Vandenberghe, 2004).

1144 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,

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

1145 where $\nabla_{\boldsymbol{\theta}} L$ is the gradient computed over the entire training set, and $\eta^{(t)}$ is the **step size**
1146 at iteration t . If the objective L is a convex function of $\boldsymbol{\theta}$, then this procedure is guaranteed
1147 to terminate at the global optimum, for appropriate schedule of learning rates, $\eta^{(t)}$.¹⁶

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

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

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

1148 In practice, gradient descent can be slow to converge, as the gradient can become
 1149 infinitesimally small. Faster convergence can be obtained by second-order Newton opti-
 1150 mization, which incorporates the inverse of the **Hessian matrix**,

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

1151 The size of the Hessian matrix is quadratic in the number of features. In the bag-of-words
 1152 representation, this is usually too big to store, let alone invert. **Quasi-Network optimiza-**
 1153 **tion** techniques maintain a low-rank approximation to the inverse of the Hessian matrix.
 1154 Such techniques usually converge more quickly than gradient descent, while remaining
 1155 computationally tractable even for large feature sets. A popular quasi-Newton algorithm
 1156 is **L-BFGS** (Liu and Nocedal, 1989), which is implemented in many scientific computing
 1157 environments, such as `scipy` and `Matlab`.

1158 For any gradient-based technique, the user must set the learning rates $\eta^{(t)}$. While con-
 1159 vergence proofs usually employ a decreasing learning rate, in practice, it is common to fix
 1160 $\eta^{(t)}$ to a small constant, like 10^{-3} . The specific constant can be chosen by experimentation,
 1161 although there is research on determining the learning rate automatically (Schaul et al.,
 1162 2013; Wu et al., 2018).

1163 2.5.2 Online optimization

1164 Batch optimization computes the objective on the entire training set before making an up-
 1165 date. This may be inefficient, because at early stages of training, a small number of train-
 1166 ing examples could point the learner in the correct direction. **Online learning** algorithms
 1167 make updates to the weights while iterating through the training data. The theoretical
 1168 basis for this approach is a stochastic approximation to the true objective function,

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

1169 where the instance $(\mathbf{x}^{(j)}, y^{(j)})$ is sampled at random from the full dataset.

1170 In **stochastic gradient descent**, the approximate gradient is computed by randomly
 1171 sampling a single instance, and an update is made immediately. This is similar to the
 1172 perceptron algorithm, which also updates the weights one instance at a time. In **mini-**
 1173 **batch** stochastic gradient descent, the gradient is computed over a small set of instances.
 1174 A typical approach is to set the minibatch size so that the entire batch fits in memory on a
 1175 graphics processing unit (GPU; Neubig et al., 2017). It is then possible to speed up learn-
 1176 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  $\boldsymbol{\theta}^{(t)}$ 

```

1177 Algorithm 5 offers a generalized view of gradient descent. In standard gradient de-
 1178 scent, the batcher returns a single batch with all the instances. In stochastic gradient de-
 1179 scent, it returns N batches with one instance each. In mini-batch settings, the batcher
 1180 returns B minibatches, $1 < B < N$.

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

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

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

1181 where j iterates over features in $\mathbf{f}(\mathbf{x}, y)$.

1182 In most cases, the number of active features for any instance is much smaller than the
 1183 number of weights. If so, the computation cost of online optimization will be dominated
 1184 by the update from the regularization term, $\lambda \boldsymbol{\theta}$. The solution is to be “lazy”, updating
 1185 each θ_j only as it is used. To implement lazy updating, store an additional parameter τ_j ,
 1186 which is the iteration at which θ_j was last updated. If θ_j is needed at time t , the $t - \tau$

1187 regularization updates can be performed all at once. This strategy is described in detail
 1188 by Kummerfeld et al. (2015).

1189 2.6 *Additional topics in classification

1190 Throughout this text, advanced topics will be marked with an asterisk.

1191 2.6.1 Feature selection by regularization

1192 In logistic regression and large-margin classification, generalization can be improved by
 1193 regularizing the weights towards 0, using the L_2 norm. But rather than encouraging
 1194 weights to be small, it might be better for the model to be **sparse**: it should assign weights
 1195 of exactly zero to most features, and only assign non-zero weights to features that are
 1196 clearly necessary. This idea can be formalized by the L_0 norm, $L_0 = \|\theta\|_0 = \sum_j \delta(\theta_j \neq 0)$,
 1197 which applies a constant penalty for each non-zero weight. This norm can be thought
 1198 of as a form of **feature selection**: optimizing the L_0 -regularized conditional likelihood is
 1199 equivalent to trading off the log-likelihood against the number of active features. Reduc-
 1200 ing the number of active features is desirable because the resulting model will be fast,
 1201 low-memory, and should generalize well, since irrelevant features will be pruned away.
 1202 Unfortunately, the L_0 norm is non-convex and non-differentiable. Optimization under L_0
 1203 regularization is **NP-hard**, meaning that it can be solved efficiently only if P=NP (Ge et al.,
 1204 2011).

1205 A useful alternative is the L_1 norm, which is equal to the sum of the absolute values
 1206 of the weights, $\|\theta\|_1 = \sum_j |\theta_j|$. The L_1 norm is convex, and can be used as an approxima-
 1207 tion to L_0 (Tibshirani, 1996). Conveniently, the L_1 norm also performs feature selection,
 1208 by driving many of the coefficients to zero; it is therefore known as a **sparsity inducing**
 1209 **regularizer**. The L_1 norm does not have a gradient at $\theta_j = 0$, so we must instead optimize
 1210 the L_1 -regularized objective using **subgradient** methods. The associated stochastic sub-
 1211 gradient descent algorithms are only somewhat more complex than conventional SGD;
 1212 Sra et al. (2012) survey approaches for estimation under L_1 and other regularizers.

1213 Gao et al. (2007) compare L_1 and L_2 regularization on a suite of NLP problems, finding
 1214 that L_1 regularization generally gives similar accuracy to L_2 regularization, but that L_1
 1215 regularization produces models that are between ten and fifty times smaller, because more
 1216 than 90% of the feature weights are set to zero.

1217 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**,

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

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

1218 This is the origin of the name **logistic regression**. Logistic regression can be viewed as
 1219 part of a larger family of **generalized linear models** (GLMs), in which various other “link
 1220 functions” convert between the inner product $\boldsymbol{\theta} \cdot \mathbf{x}$ and the parameter of a conditional
 1221 probability distribution.

1222 In the early NLP literature, logistic regression is frequently called **maximum entropy**
 1223 classification (Berger et al., 1996). This name refers to an alternative formulation, in
 1224 which the goal is to find the maximum entropy probability function that satisfies **moment-**
 1225 **matching** constraints. These constraints specify that the empirical counts of each feature
 1226 should match the expected counts under the induced probability distribution $p_{Y|X;\boldsymbol{\theta}}$.

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

1227 The moment-matching constraint is satisfied exactly when the derivative of the condi-
 1228 tional log-likelihood function (Equation 2.64) is equal to zero. However, the constraint
 1229 can be met by many values of $\boldsymbol{\theta}$, so which should we choose?

1230 The **entropy** of the conditional probability distribution $p_{Y|X}$ is,

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

1231 where \mathcal{X} is the set of all possible feature vectors, and $p_X(\mathbf{x})$ is the probability of observing
 1232 the base features \mathbf{x} . The distribution p_X is unknown, but it can be estimated by summing
 1233 over all the instances in the training set,

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

1234 If the entropy is large, the likelihood function is smooth across possible values of y ;
 1235 if it is small, the likelihood function is sharply peaked at some preferred value; in the
 1236 limiting case, the entropy is zero if $p(y | x) = 1$ for some y . The maximum-entropy cri-
 1237 terion chooses to make the weakest commitments possible, while satisfying the moment-
 1238 matching constraints from Equation 2.78. The solution to this constrained optimization
 1239 problem is identical to the maximum conditional likelihood (logistic-loss) formulation
 1240 that was presented in § 2.4.

1241 2.7 Summary of learning algorithms

1242 It is natural to ask which learning algorithm is best, but the answer depends on what
 1243 characteristics are important to the problem you are trying to solve.

1244 **Naïve Bayes** *Pros:* easy to implement; estimation is fast, requiring only a single pass over
 1245 the data; assigns probabilities to predicted labels; controls overfitting with smoothing
 1246 parameter. *Cons:* often has poor accuracy, especially with correlated features.

1247 **Perceptron** *Pros:* easy to implement; online; error-driven learning means that accuracy
 1248 is typically high, especially after averaging. *Cons:* not probabilistic; hard to know
 1249 when to stop learning; lack of margin can lead to overfitting.

1250 **Support vector machine** *Pros:* optimizes an error-based metric, usually resulting in high
 1251 accuracy; overfitting is controlled by a regularization parameter. *Cons:* not proba-
 1252 bilistic.

1253 **Logistic regression** *Pros:* error-driven and probabilistic; overfitting is controlled by a reg-
 1254 ularization parameter. *Cons:* batch learning requires black-box optimization; logistic
 1255 loss can “overtrain” on correctly labeled examples.

1256 One of the main distinctions is whether the learning algorithm offers a probability
 1257 over labels. This is useful in modular architectures, where the output of one classifier
 1258 is the input for some other system. In cases where probability is not necessary, the sup-
 1259 port vector machine is usually the right choice, since it is no more difficult to implement
 1260 than the perceptron, and is often more accurate. When probability is necessary, logistic
 1261 regression is usually more accurate than Naïve Bayes.

1262 Additional resources

1263 For more on classification, you can consult a textbook on machine learning (e.g., Mur-
 1264 phy, 2012), although the notation will differ slightly from what is typical in natural lan-
 1265 guage processing. Probabilistic methods are surveyed by Hastie et al. (2009), and Mohri
 1266 et al. (2012) emphasize theoretical considerations. Online learning is a rapidly moving
 1267 subfield of machine learning, and Bottou et al. (2016) describes progress through 2016.
 1268 Kummerfeld et al. (2015) empirically review several optimization algorithms for large-
 1269 margin learning. The python toolkit `scikit-learn` includes implementations of all of
 1270 the algorithms described in this chapter (Pedregosa et al., 2011).

1271 **Exercises**

- 1272 1. Let \mathbf{x} be a bag-of-words vector such that $\sum_{j=1}^V x_j = 1$. Verify that the multinomial
 1273 probability $p_{\text{mult}}(\mathbf{x}; \phi)$, as defined in Equation 2.12, is identical to the probability of
 1274 the same document under a categorical distribution, $p_{\text{cat}}(\mathbf{w}; \phi)$.
- 1275 2. Derive the maximum-likelihood estimate for the parameter μ in Naïve Bayes.
- 1276 3. As noted in the discussion of averaged perceptron in § 2.2.2, the computation of the
 1277 running sum $\mathbf{m} \leftarrow \mathbf{m} + \boldsymbol{\theta}$ is unnecessarily expensive, requiring $K \times V$ operations.
 1278 Give an alternative way to compute the averaged weights $\bar{\boldsymbol{\theta}}$, with complexity that is
 1279 independent of V and linear in the sum of feature sizes $\sum_{i=1}^N |\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)})|$.
- 1280 4. Consider a dataset that is comprised of two identical instances $\mathbf{x}^{(1)} = \mathbf{x}^{(2)}$ with
 1281 distinct labels $y^{(1)} \neq y^{(2)}$. Assume all features are binary $x_j \in \{0, 1\}$ for all j .

1282 Now suppose that the averaged perceptron always chooses $i = 1$ when t is even,
 1283 and $i = 2$ when t is odd, and that it will terminate under the following condition:

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

1284 In words, the algorithm stops when the largest change in the averaged weights is
 1285 less than or equal to ϵ . Compute the number of iterations before the averaged per-
 1286 ceptron terminates.

- 1287 5. Suppose you have two labeled datasets D_1 and D_2 , with the same features and la-
 1288 bels.
- 1289 • Let $\boldsymbol{\theta}^{(1)}$ be the unregularized logistic regression (LR) coefficients from training
 1290 on dataset D_1 .
 - 1291 • Let $\boldsymbol{\theta}^{(2)}$ be the unregularized LR coefficients (same model) from training on
 1292 dataset D_2 .
 - 1293 • Let $\boldsymbol{\theta}^*$ be the unregularized LR coefficients from training on the combined
 1294 dataset $D_1 \cup D_2$.

Under these conditions, prove that for any feature j ,

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

1295

1296 **Chapter 3**

1297 **Nonlinear classification**

1298 Linear classification may seem like all we need for natural language processing. The bag-
1299 of-words representation is inherently high dimensional, and the number of features is
1300 often larger than the number of training instances. This means that it is usually possible
1301 to find a linear classifier that perfectly fits the training data. Moving to nonlinear classifi-
1302 cation may therefore only increase the risk of overfitting. For many tasks, **lexical features**
1303 (words) are meaningful in isolation, and can offer independent evidence about the in-
1304 stance label — unlike computer vision, where individual pixels are rarely informative,
1305 and must be evaluated holistically to make sense of an image. For these reasons, natu-
1306 ral language processing has historically focused on linear classification to a greater extent
1307 than other machine learning application domains.

1308 But in recent years, nonlinear classifiers have swept through natural language pro-
1309 cessing, and are now the default approach for many tasks (Manning, 2016). There are at
1310 least three reasons for this change.

- 1311 • There have been rapid advances in **deep learning**, a family of nonlinear meth-
1312 ods that learn complex functions of the input through multiple layers of computa-
1313 tion (Goodfellow et al., 2016).
- 1314 • Deep learning facilitates the incorporation of **word embeddings**, which are dense
1315 vector representations of words. Word embeddings can be learned from large amounts
1316 of unlabeled data, and enable generalization to words that do not appear in the an-
1317notated training data (word embeddings are discussed in detail in chapter 14).
- 1318 • A third reason for the rise of deep nonlinear learning algorithms is hardware. Many
1319 deep learning models can be implemented efficiently on graphics processing units
1320 (GPUs), offering substantial performance improvements over CPU-based comput-
1321 ing.

1322 This chapter focuses on **neural networks**, which are the dominant approach for non-

1323 linear classification in natural language processing today.¹ Historically, a few other non-
 1324 linear learning methods have been applied to language data:

- 1325 • **Kernel methods** are generalizations of the **nearest-neighbor** classification rule, which
 1326 classifies each instance by the label of the most similar example in the training
 1327 set (Hastie et al., 2009). The application of the **kernel support vector machine** to
 1328 information extraction is described in chapter 17.
- 1329 • **Decision trees** classify instances by checking a set of conditions. Scaling decision
 1330 trees to bag-of-words inputs is difficult, but decision trees have been successful in
 1331 problems such as coreference resolution (chapter 15), where more compact feature
 1332 sets can be constructed (Soon et al., 2001).
- 1333 • **Boosting** and related **ensemble methods** work by combining the predictions of sev-
 1334 eral “weak” classifiers, each of which may consider only a small subset of features.
 1335 Boosting has been successfully applied to text classification (Schapire and Singer,
 1336 2000) and syntactic analysis (Abney et al., 1999), and remains one of the most suc-
 1337 cessful methods on machine learning competition sites such as Kaggle (Chen and
 1338 Guestrin, 2016).

1339 3.1 Feedforward neural networks

1340 Consider the problem of building a classifier for movie reviews. The goal is to predict
 1341 a label $y \in \{\text{GOOD}, \text{BAD}, \text{OKAY}\}$ from a representation of the text of each document, x .
 1342 But what makes a good movie? The story, acting, cinematography, soundtrack, and so
 1343 on. Now suppose the training set contains labels for each of these additional features,
 1344 $z = [z_1, z_2, \dots, z_{K_z}]^\top$. With such information, we could build a two-step classifier:

- 1345 1. **Use the text x to predict the features z .** Specifically, train a logistic regression clas-
 1346 sifier to compute $p(z_k | x)$, for each $k \in \{1, 2, \dots, K_z\}$.
- 1347 2. **Use the features z to predict the label y .** Again, train a logistic regression classifier
 1348 to compute $p(y | z)$. On test data, z is unknown, so we use the probabilities $p(z | x)$
 1349 from the first layer as the features.

1350 This setup is shown in Figure 3.1, which describes the proposed classifier in a **compu-**
 1351 **tation graph**: the text features x are connected to the middle layer z , which in turn is
 1352 connected to the label y .

1353 Since each $z_k \in \{0, 1\}$, we can treat $p(z_k | x)$ as a binary classification problem, using
 1354 binary logistic regression:

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

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

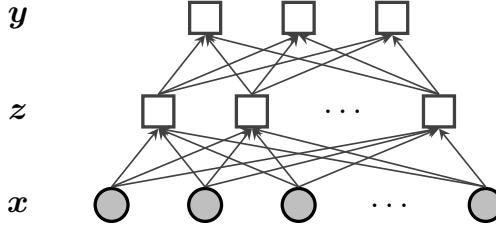


Figure 3.1: A feedforward neural network. Shaded circles indicate observed features, usually words; squares indicate nodes in the computation graph, which are computed from the information carried over the incoming arrows.

1355 where $\sigma(\cdot)$ is the **sigmoid** function (shown in Figure 3.2), and the matrix $\Theta^{(x \rightarrow z)} \in \mathbb{R}^{K_z \times V}$
 1356 is constructed by stacking the weight vectors for each z_k ,

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

1357 We will assume that x contains a term with a constant value of 1, so that a corresponding
 1358 offset parameter is included in each $\theta_k^{(x \rightarrow z)}$.

1359 The output layer is computed by the multi-class logistic regression probability,

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

1360 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
 1361 constructed by concatenation,

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

1362 The vector of probabilities over each possible value of y is denoted,

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

1363 where element j in the output of the **SoftMax** function is computed as in Equation 3.3.

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

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

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

1364 where $\sigma(\cdot)$ is now applied **elementwise** to the vector of inner products,

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

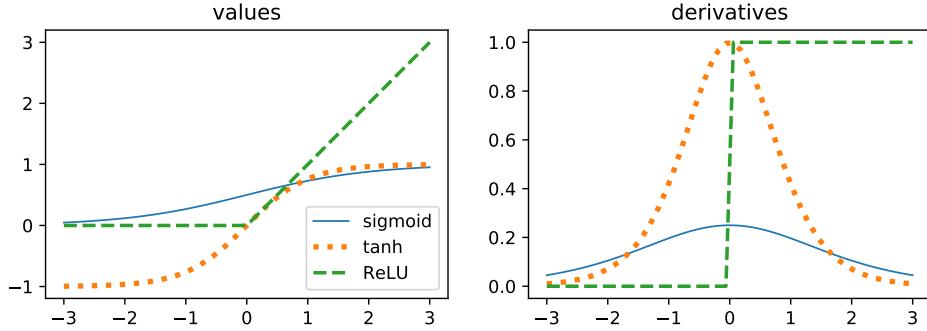


Figure 3.2: The sigmoid, tanh, and ReLU activation functions

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

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

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

1365 This defines a classification model that predicts the label $y \in \mathcal{Y}$ from the base features x ,
 1366 through a “hidden layer” z . This is a **feedforward neural network**.²

1367 3.2 Designing neural networks

1368 This feedforward neural network can be generalized in a number of ways.

1369 3.2.1 Activation functions

1370 If the hidden layer is viewed as a set of latent features, then the sigmoid function repre-
 1371 sents the extent to which each of these features is “activated” by a given input. However,
 1372 the hidden layer can be regarded more generally as a nonlinear transformation of the in-
 1373 put. This opens the door to many other activation functions, some of which are shown in
 1374 Figure 3.2. At the moment, the choice of activation functions is more art than science, but
 1375 a few points can be made about the most popular varieties:

- 1376 • The range of the sigmoid function is $(0, 1)$. The bounded range ensures that a cas-
 1377 cade of sigmoid functions will not “blow up” to a huge output, and this is impor-

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

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

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

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

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

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

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

3.2.2 Network structure

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

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

1405 It is also possible to “short circuit” a hidden layer, by propagating information directly
 1406 from the input to the next higher level of the network. This is the idea behind **residual net-**
 1407 **works**, which propagate information directly from the input to the subsequent layer (He
 1408 et al., 2016),

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

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

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

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

1409 where \odot refers to an elementwise vector product, and $\mathbf{1}$ is a column vector of ones. The
 1410 sigmoid function is applied elementwise to its input; recall that the output of this function
 1411 is restricted to the range $[0, 1]$. Gating is also used in the **long short-term memory (LSTM)**,
 1412 which is discussed in chapter 6. Residual and highway connections address a problem
 1413 with deep architectures: repeated application of a nonlinear activation function can make
 1414 it difficult to learn the parameters of the lower levels of the network, which are too distant
 1415 from the supervision signal.

1416 3.2.3 Outputs and loss functions

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

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

1417 where $\Theta = \{\Theta^{(x \rightarrow z)}, \Theta^{(z \rightarrow y)}, \mathbf{b}\}$ is the entire set of parameters.

This loss can be written alternatively as follows:

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

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

1418 where $e_{y^{(i)}}$ is a **one-hot vector** of zeros with a value of 1 at position $y^{(i)}$. The inner product
 1419 between $e_{y^{(i)}}$ and $\log \tilde{y}$ is also called the multinomial **cross-entropy**, and this terminology
 1420 is preferred in many neural networks papers and software packages.

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

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

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

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

- 1421 In regression problems, the output is a scalar or vector (see § 4.1.2). For these problems, a
 1422 typical loss function is the squared error $(y - \hat{y})^2$ or squared norm $\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2$.

1423 3.2.4 Inputs and lookup layers

1424 In text classification, the input layer \mathbf{x} can refer to a bag-of-words vector, where x_j is
 1425 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
 1426 represented by the vector $\theta_j^{(x \rightarrow z)}$. This vector is sometimes described as the **embedding** of
 1427 word j , and can be learned from unlabeled data, using techniques discussed in chapter 14.
 1428 The columns of $\Theta^{(x \rightarrow z)}$ are each K_z -dimensional word embeddings.

1429 Chapter 2 presented an alternative view of text documents, as a sequence of word
 1430 tokens, w_1, w_2, \dots, w_M . In a neural network, each word token w_m is represented with
 1431 a one-hot vector, $e_{w_m} \in \mathbb{R}^V$. The matrix-vector product $\Theta^{(x \rightarrow z)} e_{w_m}$ returns the embed-
 1432 ding of word w_m . The complete document can be represented by horizontally concatenating
 1433 these one-hot vectors, $\mathbf{W} = [e_{w_1}, e_{w_2}, \dots, e_{w_M}]$, and the bag-of-words representation can
 1434 be recovered from the matrix-vector product $\mathbf{W} \mathbf{1}$, which simply sums each row over the
 1435 tokens $m = \{1, 2, \dots, M\}$. The matrix product $\Theta^{(x \rightarrow z)} \mathbf{W}$ contains the horizontally con-
 1436 catenated embeddings of each word in the document, which will be useful as the starting
 1437 point for **convolutional neural networks** (see § 3.4). This is sometimes called a **lookup**
 1438 **layer**, because the first step is to lookup the embeddings for each word in the input text.

1439 3.3 Learning neural networks

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

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

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

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

- 1440 where f is an elementwise activation function, such as σ or ReLU.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1445 For intuition, consider each of the terms in the product.

- 1446 • 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
1447 doesn't matter how z_k is computed, and so $\frac{\partial \ell^{(i)}}{\partial \theta_{n,k}^{(x \rightarrow z)}} \rightarrow 0$.
- 1448 • If z_k is near 1 or 0, then the curve of the sigmoid function (Figure 3.2) is nearly flat,
1449 and changing the inputs will make little local difference. The term $z_k \times (1 - z_k)$ is
1450 maximized at $z_k = \frac{1}{2}$, where the slope of the sigmoid function is steepest.
- 1451 • If $x_n = 0$, then it does not matter how we set the weights $\theta_{n,k}^{(x \rightarrow z)}$, so $\frac{\partial \ell^{(i)}}{\partial \theta_{n,k}^{(x \rightarrow z)}} = 0$.

1452 3.3.1 Backpropagation

1453 In the equations above, the value $\frac{\partial \ell^{(i)}}{\partial z_k}$ is reused in the derivatives with respect to each
1454 $\theta_{n,k}^{(x \rightarrow z)}$. It should therefore be computed once, and then cached. Furthermore, we should
1455 only compute any derivative once we have already computed all of the necessary "inputs"
1456 demanded by the chain rule of differentiation. This combination of sequencing, caching,
1457 and differentiation is known as **backpropagation**. It can be generalized to any directed
1458 acyclic **computation graph**.

1459 A computation graph is a declarative representation of a computational process. At
1460 each node t , compute a value v_t by applying a function f_t to a (possibly empty) list of
1461 parent nodes, π_t . For example, in a feedforward network with one hidden layer, there are
1462 nodes for the input $\mathbf{x}^{(i)}$, the hidden layer \mathbf{z} , the predicted output $\tilde{\mathbf{y}}$, and the parameters
1463 $\{\Theta^{(x \rightarrow z)}, \Theta^{(z \rightarrow y)}, \mathbf{b}\}$. During training, there is also a node for the observed label $y^{(i)}$ and
1464 the loss $\ell^{(i)}$. Computation graphs have three main types of nodes:

1465 **Variables.** The variables include the *inputs* \mathbf{x} , the *hidden nodes* \mathbf{z} , the outputs \mathbf{y} , and the
1466 loss function. Inputs are variables that do not have parents. Backpropagation com-
1467putes the gradients with respect to all variables except the inputs, but does not up-
1468 date the variables during learning.

1469 **Parameters.** In a feedforward network, the parameters include the weights and offsets.
1470 Parameter nodes do not have parents, and they are updated during learning.

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

```

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

```

1471 **Objective.** The *objective* node is not the parent of any other node. Backpropagation begins
 1472 by computing the gradient with respect to this node.

1473 If the computation graph is a directed acyclic graph, then it is possible to order the
 1474 nodes with a topological sort, so that if node t is a parent of node t' , then $t < t'$. This
 1475 means that the values $\{v_t\}_{t=1}^T$ can be computed in a single forward pass. The topolog-
 1476 ical sort is reversed when computing gradients: each gradient g_t is computed from the
 1477 gradients of the children of t , implementing the chain rule of differentiation. The general
 1478 backpropagation algorithm for computation graphs is shown in Algorithm 6, and illus-
 1479 trated in Figure 3.3.

1480 While the gradients with respect to each parameter may be complex, they are com-
 1481 posed of products of simple parts. For many networks, all gradients can be computed
 1482 through **automatic differentiation**. This means that end users need only specify the feed-
 1483 forward computation, and the gradients necessary for learning can be obtained automati-
 1484 cally. There are many software libraries that perform automatic differentiation on compu-
 1485 tation graphs, such as Torch (Collobert et al., 2011), TensorFlow (Abadi et al., 2016), and
 1486 DyNet (Neubig et al., 2017). One important distinction between these libraries is whether
 1487 they support **dynamic computation graphs**, in which the structure of the computation
 1488 graph varies across instances. Static computation graphs are compiled in advance, and
 1489 can be applied to fixed-dimensional data, such as bag-of-words vectors. In many natu-
 1490 ral language processing problems, each input has a distinct structure, requiring a unique
 1491 computation graph.

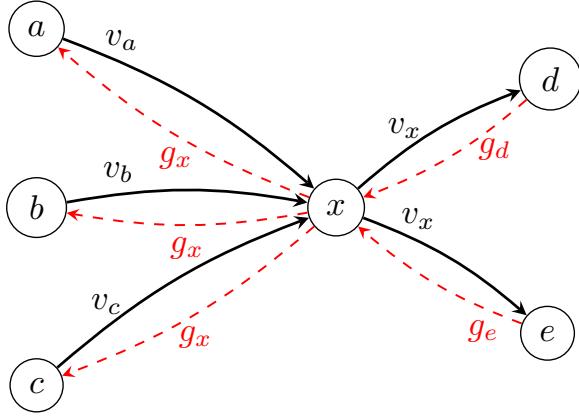


Figure 3.3: Backpropagation at a single node x in the computation graph. The values of the predecessors v_a, v_b, v_c are the inputs to x , which computes v_x , and passes it on to the successors d and e . The gradients at the successors g_d and g_e are passed back to x , where they are incorporated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

1492 3.3.2 Regularization and dropout

1493 In linear classification, overfitting was addressed by augmenting the objective with a reg-
 1494 ularization term, $\lambda \|\theta\|_2^2$. This same approach can be applied to feedforward neural net-
 1495 works, penalizing each matrix of weights:

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

1496 where $\|\Theta\|_F^2 = \sum_{i,j} \theta_{i,j}^2$ is the squared **Frobenius norm**, which generalizes the L_2 norm
 1497 to matrices. The bias parameters b are not regularized, as they do not contribute to the
 1498 sensitivity of the classifier to the inputs. In gradient-based optimization, the practical
 1499 effect of Frobenius norm regularization is that the weights “decay” towards zero at each
 1500 update, motivating the alternative name **weight decay**.

1501 Another approach to controlling model complexity is **dropout**, which involves ran-
 1502 domly setting some computation nodes to zero during training (Srivastava et al., 2014).
 1503 For example, in the feedforward network, on each training instance, with probability ρ we
 1504 set each input x_n and each hidden layer node z_k to zero. Srivastava et al. (2014) recom-
 1505 mend $\rho = 0.5$ for hidden units, and $\rho = 0.2$ for input units. Dropout is also incorpo-
 1506 rated in the gradient computation, so if node z_k is dropped, then none of the weights $\theta_k^{(x \rightarrow z)}$ will
 1507 be updated for this instance. Dropout prevents the network from learning to depend too
 1508 much on any one feature or hidden node, and prevents **feature co-adaptation**, in which a

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

3.3.3 *Learning theory

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

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

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

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

1545 stochastic gradient descent, and by feature noising techniques such as dropout, can help
 1546 online optimization to escape saddle points and find high-quality optima (Ge et al., 2015).
 1547 Other techniques address saddle points directly, using local reconstructions of the Hessian
 1548 matrix (Dauphin et al., 2014) or higher-order derivatives (Anandkumar and Ge, 2016).

1549 **3.3.4 Tricks**

1550 Getting neural networks to work effectively sometimes requires heuristic “tricks” (Bottou,
 1551 2012; Goodfellow et al., 2016; Goldberg, 2017b). This section presents some tricks that are
 1552 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]

1553 For the weights leading to a ReLU activation function, He et al. (2015) use similar argu-
 1554 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:

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

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

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

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

1555 The matrix \mathbf{U} contains the **singular vectors** of \mathbf{A} , and is guaranteed to be orthonormal.
 1556 For more on singular value decomposition, see chapter 14.

1557 Even with careful initialization, there can still be significant variance in the final re-
 1558 sults. It can be useful to make multiple training runs, and select the one with the best
 1559 performance on a heldout development set.

1560 **Clipping and normalizing the gradients** As already discussed, the magnitude of the
 1561 gradient can pose problems for learning: too large, and learning can diverge, with suc-
 1562 ccessive updates thrashing between increasingly extreme values; too small, and learning can
 1563 grind to a halt. Several heuristics have been proposed to address this issue.

1564 • In **gradient clipping** (Pascanu et al., 2013), an upper limit is placed on the norm of
 1565 the gradient, and the gradient is rescaled when this limit is exceeded,

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

1564 • In **batch normalization** (Ioffe and Szegedy, 2015), the inputs to each computation
 1565 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:

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

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

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

1566 Empirically, this speeds convergence of deep architectures. One explanation is that
 1567 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:

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

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

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

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

1568 Layer normalization has similar motivations to batch normalization, but it can be
 1569 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,

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

1570 where $g_{t,j}$ is the gradient with respect to parameter j at time t , and $\beta \in [0, 1]$. This term
 1571 places more emphasis on recent gradients, and is employed in the **AdaDelta** (Zeiler, 2012)
 1572 and **Adam** (Kingma and Ba, 2014) optimizers. Online optimization and its theoretical
 1573 background are reviewed by Bottou et al. (2016). **Early stopping**, mentioned in § 2.2.2,
 1574 can help to avoid overfitting, by terminating training after reaching a plateau in the per-
 1575 formance on a heldout validation set.

1576 3.4 Convolutional neural networks

1577 A basic weakness of the bag-of-words model is its inability to account for the ways in
 1578 which words combine to create meaning, including even simple reversals such as *not*
 1579 *pleasant, hardly a generous offer*, and *I wouldn't mind missing the flight*. Similarly, computer
 1580 vision faces the challenge of identifying the semantics of images from pixel features that
 1581 are uninformative in isolation. An earlier generation of computer vision research fo-
 1582 cused on designing *filters* to aggregate local pixel-level features into more meaningful
 1583 representations, such as edges and corners (e.g., Canny, 1987). Similarly, earlier NLP re-
 1584 search attempted to capture multiword linguistic phenomena by hand-designed lexical
 1585 patterns (Hobbs et al., 1997). In both cases, the output of the filters and patterns could

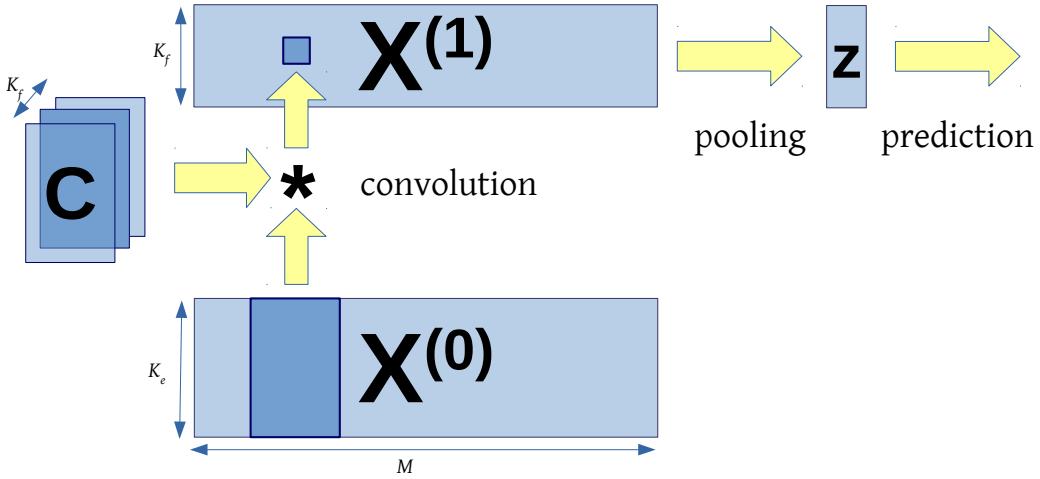


Figure 3.4: A convolutional neural network for text classification

1586 then act as base features in a linear classifier. But rather than designing these feature ex-
 1587 tractors by hand, a better approach is to learn them, using the magic of backpropagation.
 1588 This is the idea behind **convolutional neural networks**.

1589 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,

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

1590 where f is an activation function such as tanh or ReLU, and \mathbf{b} is a vector of offsets. The
 1591 convolution operation slides the matrix $\mathbf{C}^{(k)}$ across the columns of $\mathbf{X}^{(0)}$; at each position
 1592 m , compute the elementwise product $\mathbf{C}^{(k)} \odot \mathbf{X}_{m:m+h-1}^{(0)}$, and take the sum.

1593 A simple filter might compute a weighted average over nearby words,

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

(c) Jacob Eisenstein 2018. Work in progress.

1594 thereby representing trigram units like *not so unpleasant*. In **one-dimensional convolution**,
 1595 each filter matrix $\mathbf{C}^{(k)}$ is constrained to have non-zero values only at row k (Kalchbrenner et al., 2014).

1597 To deal with the beginning and end of the input, the base matrix $\mathbf{X}^{(0)}$ may be padded
 1598 with h column vectors of zeros at the beginning and end; this is known as **wide convolution**.
 1599 If padding is not applied, then the output from each layer will be $h - 1$ units smaller
 1600 than the input; this is known as **narrow convolution**. The filter matrices need not have
 1601 identical filter widths, so more generally we could write h_k to indicate width of filter
 1602 $\mathbf{C}^{(k)}$. As suggested by the notation $\mathbf{X}^{(0)}$, multiple layers of convolution may be applied,
 1603 so that $\mathbf{X}^{(d)}$ is the input to $\mathbf{X}^{(d+1)}$.

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

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

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

1604 The vector \mathbf{z} can now act as a layer in a feedforward network, culminating in a prediction
 1605 \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 :

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

1606 The computer vision literature has produced a huge variety of convolutional architectures,
 1607 and many of these bells and whistles can be applied to text data. One avenue for
 1608 improvement is more complex pooling operations, such as k -max pooling (Kalchbrenner
 1609 et al., 2014), which returns a matrix of the k largest values for each filter. Another innovation
 1610 is the use of **dilated convolution** to build multiscale representations (Yu and Koltun,
 1611 2016). At each layer, the convolutional operator applied in *strides*, skipping ahead by s
 1612 steps after each feature. As we move up the hierarchy, each layer is s times smaller than
 1613 the layer below it, effectively summarizing the input. This idea is shown in Figure 3.5.
 1614 Multi-layer convolutional networks can also be augmented with “shortcut” connections,
 1615 as in the ResNet model from § 3.2.2 (Johnson and Zhang, 2017).

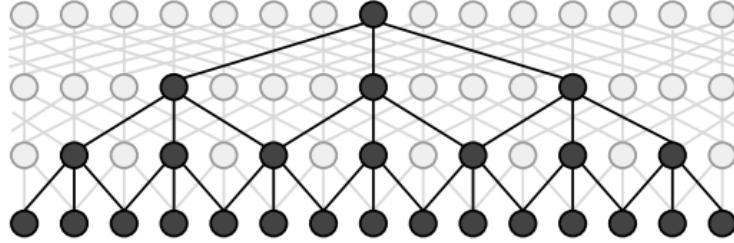


Figure 3.5: A dilated convolutional neural network captures progressively larger context through recursive application of the convolutional operator (Strubell et al., 2017) [todo: permission]

1616 Additional resources

1617 The deep learning textbook by Goodfellow et al. (2016) covers many of the topics in this
 1618 chapter in more detail. For a comprehensive review of neural networks in natural lan-
 1619 guage processing, see (Goldberg, 2017b). A seminal work on deep learning in natural
 1620 language processing is the aggressively titled “Natural Language Processing (Almost)
 1621 from Scratch”, which uses convolutional neural networks to perform a range of language
 1622 processing tasks (Collobert et al., 2011). This chapter focuses on feedforward and con-
 1623 volutional neural networks, but recurrent neural networks are one of the most important
 1624 deep learning architectures for natural language processing. They are covered extensively
 1625 in chapters 6 and 7.

1626 The role of deep learning in natural language processing research has caused angst
 1627 in some parts of the natural language processing research community (e.g., Goldberg,
 1628 2017a), especially as some of the more zealous deep learning advocates have argued that
 1629 end-to-end learning from “raw” text can eliminate the need for linguistic constructs such
 1630 as sentences, phrases, and even words (Zhang et al., 2015, originally titled *Text understand-
 1631 ing from scratch*). These developments were surveyed by Manning (2016).

1632 Exercises

- 1633 1. Prove that the softmax and sigmoid functions are equivalent when the number of
 1634 possible labels is two. Specifically, for any $\Theta^{(z \rightarrow y)}$ (omitting the offset b for sim-
 1635 plicity), show how to construct a vector of weights θ such that,

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

(c) Jacob Eisenstein 2018. Work in progress.

- 1636 2. Design a feedforward network to compute the XOR function:

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

1637 Your network should have a single output node which uses the Sign activation function.
 1638 Use a single hidden layer, with ReLU activation functions. Describe all weights
 1639 and offsets.

- 1640 3. Consider the same network as above (with ReLU activations for the hidden layer),
 1641 with an arbitrary differentiable loss function $\ell(y^{(i)}, \tilde{y})$, where \tilde{y} is the activation of
 1642 the output node. Suppose all weights and offsets are initialized to zero. Prove that
 1643 gradient-based optimization cannot learn the desired function from this initializa-
 1644 tion.
- 1645 4. The simplest solution to the previous problem relies on the use of the ReLU activa-
 1646 tion function at the hidden layer. Now consider a network with arbitrary activations
 1647 on the hidden layer. Show that if the initial weights are any uniform constant, then
 1648 it is not possible to learn the desired function.
- 1649 5. Consider a network in which: the base features are all binary, $\mathbf{x} \in \{0, 1\}^M$; the
 1650 hidden layer activation function is sigmoid, $z_k = \sigma(\theta_k \cdot \mathbf{x})$; and the initial weights
 1651 are sampled independently from a standard normal distribution, $\theta_{j,k} \sim N(0, 1)$.
- 1652 • Show how the probability of a small initial gradient on any weight, $\frac{\partial z_k}{\partial \theta_{j,k}} < \alpha$,
 1653 depends on the size of the input M . **Hint:** use the lower bound,
- $$\Pr(\sigma(\theta_k \cdot \mathbf{x}) \times (1 - \sigma(\theta_k \cdot \mathbf{x})) < \alpha) \geq 2 \Pr(\sigma(\theta_k \cdot \mathbf{x}) < \alpha), \quad [3.60]$$
- 1654 and relate this probability to the variance $V[\theta_k \cdot \mathbf{x}]$.
- 1655 • Design an alternative initialization that removes this dependence.
- 1656 6. Suppose that the parameters $\Theta = \{\Theta^{(x \rightarrow z)}, \Theta(z \rightarrow y), \mathbf{b}\}$ are a local optimum of a
 1657 feedforward network in the following sense: there exists some $\epsilon > 0$ such that,

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

1658 Define the function π as a permutation on the hidden units, as described in § 3.3.3,
 1659 so that for any Θ , $L(\Theta) = L(\Theta_\pi)$. Prove that if a feedforward network has a local
 optimum in the sense of Equation 3.61, then its loss is not a convex function of the
 parameters Θ , using the definition of convexity from § 2.3

1660 Chapter 4

1661 Linguistic applications of 1662 classification

1663 Having learned some techniques for classification, this chapter shifts the focus from math-
1664 ematics to linguistic applications. Later in the chapter, we will consider the design deci-
1665 sions involved in text classification, as well as evaluation practices.

1666 4.1 Sentiment and opinion analysis

1667 A popular application of text classification is to automatically determine the **sentiment**
1668 or **opinion polarity** of documents such as product reviews and social media posts. For
1669 example, marketers are interested to know how people respond to advertisements, ser-
1670 vices, and products (Hu and Liu, 2004); social scientists are interested in how emotions
1671 are affected by phenomena such as the weather (Hannak et al., 2012), and how both opin-
1672 ions and emotions spread over social networks (Coviello et al., 2014; Miller et al., 2011).
1673 In the field of **digital humanities**, literary scholars track plot structures through the flow
1674 of sentiment across a novel (Jockers, 2015).¹

1675 Sentiment analysis can be framed as a direct application of document classification,
1676 assuming reliable labels can be obtained. In the simplest case, sentiment analysis is a
1677 two or three-class problem, with sentiments of POSITIVE, NEGATIVE, and possibly NEU-
1678 TRAL. Such annotations could be annotated by hand, or obtained automatically through
1679 a variety of means:

- 1680 • Tweets containing happy emoticons can be marked as positive, sad emoticons as
1681 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).

- 1682 • Reviews with four or more stars can be marked as positive, two or fewer stars as
1683 negative (Pang et al., 2002).
- 1684 • Statements from politicians who are voting for a given bill are marked as positive
1685 (towards that bill); statements from politicians voting against the bill are marked as
1686 negative (Thomas et al., 2006).

1687 The bag-of-words model is a good fit for sentiment analysis at the document level: if
1688 the document is long enough, we would expect the words associated with its true senti-
1689 ment to overwhelm the others. Indeed, **lexicon-based sentiment analysis** avoids machine
1690 learning altogether, and classifies documents by counting words against positive and neg-
1691 ative sentiment word lists (Taboada et al., 2011).

1692 Lexicon-based classification is less effective for short documents, such as single-sentence
1693 reviews or social media posts. In these documents, linguistic issues like **negation** and **ir-**
1694 **realis** (Polanyi and Zaenen, 2006) — events that are hypothetical or otherwise non-factual
1695 — can make bag-of-words classification ineffective. Consider the following examples:

- 1696 (4.1) That's not bad for the first day.
- 1697 (4.2) This is not the worst thing that can happen.
- 1698 (4.3) It would be nice if you acted like you understood.
- 1699 (4.4) There is no reason at all to believe that the polluters are suddenly going to be-
1700 come reasonable. (Wilson et al., 2005)
- 1701 (4.5) This film should be brilliant. The actors are first grade. Stallone plays a happy,
1702 wonderful man. His sweet wife is beautiful and adores him. He has a fascinat-
1703 ing gift for living life fully. It sounds like a great plot, **however**, the film is a
1704 failure. (Pang et al., 2002)

1705 A minimal solution is to move from a bag-of-words model to a bag-of-**bigrams** model,
1706 where each base feature is a pair of adjacent words, e.g.,

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

1707 Bigrams can handle relatively straightforward cases, such as when an adjective is immedi-
1708 ately negated; trigrams would be required to extend to larger contexts (e.g., *not the worst*).
1709 But this approach will not scale to more complex examples like (4.4) and (4.5). More
1710 sophisticated solutions try to account for the syntactic structure of the sentence (Wilson
1711 et al., 2005; Socher et al., 2013), or apply more complex classifiers such as **convolutional**
1712 **neural networks** (Kim, 2014), which are described in chapter 3.

1713 **4.1.1 Related problems**

1714 **Subjectivity** Closely related to sentiment analysis is **subjectivity detection**, which re-
1715 quires identifying the parts of a text that express subjective opinions, as well as other non-
1716 factual content such as speculation and hypotheticals (Riloff and Wiebe, 2003). This can be
1717 done by treating each sentence as a separate document, and then applying a bag-of-words
1718 classifier: indeed, Pang and Lee (2004) do exactly this, using a training set consisting of
1719 (mostly) subjective sentences gathered from movie reviews, and (mostly) objective sen-
1720 tences gathered from plot descriptions. They augment this bag-of-words model with a
1721 graph-based algorithm that encourages nearby sentences to have the same subjectivity
1722 label.

1723 **Stance classification** In debates, each participant takes a side: for example, advocating
1724 for or against proposals like adopting a vegetarian lifestyle or mandating free college ed-
1725 ucation. The problem of stance classification is to identify the author’s position from the
1726 text of the argument. In some cases, there is training data available for each position,
1727 so that standard document classification techniques can be employed. In other cases, it
1728 suffices to classify each document as whether it is in support or opposition of the argu-
1729 ment advanced by a previous document (Anand et al., 2011). In the most challenging
1730 case, there is no labeled data for any of the stances, so the only possibility is group docu-
1731 ments that advocate the same position (Somasundaran and Wiebe, 2009). This is a form
1732 of **unsupervised learning**, discussed in chapter 5.

1733 **Targeted sentiment analysis** The expression of sentiment is often more nuanced than a
1734 simple binary label. Consider the following examples:

1735 (4.6) The vodka was good, but the meat was rotten.

1736 (4.7) Go to Heaven for the climate, Hell for the company. –Mark Twain

1737 These statements display a mixed overall sentiment: positive towards some entities (e.g.,
1738 *the vodka*), negative towards others (e.g., *the meat*). **Targeted sentiment analysis** seeks to
1739 identify the writer’s sentiment towards specific entities (Jiang et al., 2011). This requires
1740 identifying the entities in the text and linking them to specific sentiment words — much
1741 more than we can do with the classification-based approaches discussed thus far. For
1742 example, Kim and Hovy (2006) analyze sentence-internal structure to determine the topic
1743 of each sentiment expression.

1744 **Aspect-based opinion mining** seeks to identify the sentiment of the author of a review
1745 towards predefined aspects such as PRICE and SERVICE, or, in the case of (4.7), CLIMATE
1746 and COMPANY (Hu and Liu, 2004). If the aspects are not defined in advance, it may again
1747 be necessary to employ **unsupervised learning** methods to identify them (e.g., Branavan
1748 et al., 2009).

1749 **Emotion classification** While sentiment analysis is framed in terms of positive and neg-
 1750 ative categories, psychologists generally regard **emotion** as more multifaceted. For ex-
 1751 ample, Ekman (1992) argues that there are six basic emotions — happiness, surprise, fear,
 1752 sadness, anger, and contempt — and that they are universal across human cultures. Alm
 1753 et al. (2005) build a linear classifier for recognizing the emotions expressed in children’s
 1754 stories. The ultimate goal of this work was to improve text-to-speech synthesis, so that
 1755 stories could be read with intonation that reflected the emotional content. They used bag-
 1756 of-words features, as well as features capturing the story type (e.g., jokes, folktales), and
 1757 structural features that reflect the position of each sentence in the story. The task is diffi-
 1758 cult: even human annotators frequently disagreed with each other, and the best classifiers
 1759 achieved accuracy between 60-70%.

1760 4.1.2 Alternative approaches to sentiment analysis

1761 **Regression** A more challenging version of sentiment analysis is to determine not just
 1762 the class of a document, but its rating on a numerical scale (Pang and Lee, 2005). If the
 1763 scale is continuous, it is most natural to apply **regression**, identifying a set of weights θ
 1764 that minimize the squared error of a predictor $\hat{y} = \theta \cdot x + b$, where b is an offset. This
 1765 approach is called **linear regression**, and sometimes **least squares**, because the regression
 1766 coefficients θ are determined by minimizing the squared error, $(y - \hat{y})^2$. If the weights are
 1767 regularized using a penalty $\lambda \|\theta\|_2^2$, then it is **ridge regression**. Unlike logistic regression,
 1768 both linear regression and ridge regression can be solved in closed form as a system of
 1769 linear equations.

1770 **Ordinal ranking** In many problems, the labels are ordered but discrete: for example,
 1771 product reviews are often integers on a scale of 1 – 5, and grades are on a scale of A – F.
 1772 Such problems can be solved by discretizing the score $\theta \cdot x$ into “ranks”,

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

1773 where $\mathbf{b} = [b_1 = -\infty, b_2, b_3, \dots, b_K]$ is a vector of boundaries. It is possible to learn the
 1774 weights and boundaries simultaneously, using a perceptron-like algorithm (Crammer and
 1775 Singer, 2001).

1776 **Lexicon-based classification** Sentiment analysis is one of the only NLP tasks where
 1777 hand-crafted feature weights are still widely employed. In **lexicon-based classification** (Taboada
 1778 et al., 2011), the user creates a list of words for each label, and then classifies each docu-
 1779 ment based on how many of the words from each list are present. In our linear classifica-
 1780 tion framework, this is equivalent to choosing the following weights:

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

(c) Jacob Eisenstein 2018. Work in progress.

1781 where \mathcal{L}_y is the lexicon for label y . Compared to the machine learning classifiers discussed
 1782 in the previous chapters, lexicon-based classification may seem primitive. However, su-
 1783 pervised machine learning relies on large annotated datasets, which are time-consuming
 1784 and expensive to produce. If the goal is to distinguish two or more categories in a new
 1785 domain, it may be simpler to start by writing down a list of words for each category.

1786 An early lexicon was the *General Inquirer* (Stone, 1966). Today, popular sentiment lex-
 1787 cons include sentiwordnet (Esuli and Sebastiani, 2006) and an evolving set of lexicons
 1788 from Liu (2015). For emotions and more fine-grained analysis, *Linguistic Inquiry and Word*
 1789 *Count* (LIWC) provides a set of lexicons (Tausczik and Pennebaker, 2010). The MPQA lex-
 1790 icon indicates the polarity (positive or negative) of 8221 terms, as well as whether they are
 1791 strongly or weakly subjective (Wiebe et al., 2005). A comprehensive comparison of senti-
 1792 ment lexicons is offered by Ribeiro et al. (2016). Given an initial **seed lexicon**, it is possible
 1793 to automatically expand the lexicon by looking for words that frequently co-occur with
 1794 words in the seed set (Hatzivassiloglou and McKeown, 1997; Qiu et al., 2011).

1795 4.2 Word sense disambiguation

1796 Consider the the following headlines:

- 1797 (4.8) Iraqi head seeks arms
- 1798 (4.9) Prostitutes appeal to Pope
- 1799 (4.10) Drunk gets nine years in violin case²

1800 These headlines are ambiguous because they contain words that have multiple mean-
 1801 ings, or **senses**. Word sense disambiguation is the problem of identifying the intended
 1802 sense of each word token in a document. Word sense disambiguation is part of a larger
 1803 field of research called **lexical semantics**, which is concerned with meanings of the words.

1804 At a basic level, the problem of word sense disambiguation is to identify the correct
 1805 sense for each word token in a document. Part-of-speech ambiguity (e.g., noun versus
 1806 verb) is usually considered to be a different problem, to be solved at an earlier stage.
 1807 From a linguistic perspective, senses are not properties of words, but of **lemmas**, which
 1808 are canonical forms that stand in for a set of inflected words. For example, *arm*/N is a
 1809 lemma that includes the inflected form *arms*/N — the /N indicates that it we are refer-
 1810 ring to the noun, and not its **homonym** *arm*/V, which is another lemma that includes
 1811 the inflected verbs (*arm*/V, *arms*/V, *armed*/V, *arming*/V). Therefore, word sense disam-
 1812 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>

1813 and then choosing the correct sense from the inventory associated with the corresponding
 1814 lemma.³ (Part-of-speech tagging is discussed in § 8.1.)

1815 **4.2.1 How many word senses?**

1816 Words sometimes have many more than two senses, as exemplified by the word *serve*:

- 1817 • [FUNCTION]: *The tree stump served as a table*
- 1818 • [CONTRIBUTE TO]: *His evasive replies only served to heighten suspicion*
- 1819 • [PROVIDE]: *We serve only the rawest fish*
- 1820 • [ENLIST]: *She served in an elite combat unit*
- 1821 • [JAIL]: *He served six years for a crime he didn't commit*
- 1822 • [LEGAL]: *They were served with subpoenas*⁴

1823 These sense distinctions are annotated in **WordNet** (<http://wordnet.princeton.edu>), a lexical semantic database for English. WordNet consists of roughly 100,000 **synsets**,
 1824 which are groups of lemmas (or phrases) that are synonymous. An example synset is
 1825 {*chump*¹, *fool*², *sucker*¹, *mark*⁹}, where the superscripts index the sense of each lemma that
 1826 is included in the synset: for example, there are at least eight other senses of *mark* that
 1827 have different meanings, and are not part of this synset. A lemma is **polysemous** if it
 1828 participates in multiple synsets.

1829 WordNet defines the scope of the word sense disambiguation problem, and, more
 1830 generally, formalizes lexical semantic knowledge of English. (WordNets have been cre-
 1831 ated for a few dozen other languages, at varying levels of detail.) Some have argued
 1832 that WordNet's sense granularity is too fine (Ide and Wilks, 2006); more fundamentally,
 1833 the premise that word senses can be differentiated in a task-neutral way has been criti-
 1834 cized as linguistically naïve (Kilgarriff, 1997). One way of testing this question is to ask
 1835 whether people tend to agree on the appropriate sense for example sentences: accord-
 1836 ing to Mihalcea et al. (2004), people agree on roughly 70% of examples using WordNet
 1837 senses; far better than chance, but less than agreement on other tasks, such as sentiment
 1838 annotation (Wilson et al., 2005).

1839 ***Other lexical semantic relations** Besides **synonymy**, WordNet also describes many
 1840 other lexical semantic relationships, including:

- 1841 • **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).

- 1843 • **hyponymy:** x is a special case of y , e.g. RED-COLOR; the inverse relationship is
 1844 **hyperonymy**;
 1845 • **meronymy:** x is a part of y , e.g., WHEEL-BICYCLE; the inverse relationship is **holonymy**.

1846 Classification of these relations can be performed by searching for character-
 1847 istic patterns between pairs of words, e.g., X , *such as* Y , which signals hyponymy (Hearst,
 1848 1992), or X *but* Y , which signals antonymy (Hatzivassiloglou and McKeown, 1997). An-
 1849 other approach is to analyze each term's **distributional statistics** (the frequency of its
 1850 neighboring words). Such approaches are described in detail in chapter 14.

1851 **4.2.2 Word sense disambiguation as classification**

1852 How can we tell living *plants* from manufacturing *plants*? The context is often critical:

- 1853 (4.11) Town officials are hoping to attract new manufacturing plants through weakened
 1854 environmental regulations.
 1855 (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 treat-
 ing each context as a pseudo-document. The feature function is then,

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

1856 As in document classification, many of these features are irrelevant, but a few are very
 1857 strong predictors. In this example, the context word *endangered* is a strong signal that
 1858 the intended sense is biology rather than manufacturing. We would therefore expect a
 1859 learning algorithm to assign high weight to (*endangered*, BIOLOGY), and low weight to
 1860 (*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.,

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

1861 These are called **collocation features**, and they give more information about the specific
 1862 role played by each context word. This idea can be taken further by incorporating addi-
 1863 tional syntactic information about the grammatical role played by each context feature,
 1864 such as the **dependency path** (see chapter 11).

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

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

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

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

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

4.3 Design decisions for text classification

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

4.3.1 What is a word?

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

4.3.1.1 Tokenization

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

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

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

1898 to define a subset of characters as whitespace, and then split the text on these tokens.
 1899 However, whitespace-based tokenization is not ideal: we may want to split conjunctions
 1900 like *isn't* and hyphenated phrases like *prize-winning* and *half-asleep*, and we likely want
 1901 to separate words from commas and periods that immediately follow them. At the same
 1902 time, it would be better not to split abbreviations like *U.S.* and *Ph.D.* In languages with
 1903 Roman scripts, tokenization is typically performed using regular expressions, with mod-
 1904 ules designed to handle each of these cases. For example, the `nltk` package includes a
 1905 number of tokenizers (Loper and Bird, 2002); the outputs of four of the better-known tok-
 1906 enizers are shown in Figure 4.1. Social media researchers have found that emoticons and
 1907 other forms of orthographic variation pose new challenges for tokenization, leading to the
 1908 development of special purpose tokenizers to handle these phenomena (O'Connor et al.,
 1909 2010).

1910 Tokenization is a language-specific problem, and each language poses unique chal-
 1911 lenges. For example, Chinese does not include spaces between words, nor any other
 1912 consistent orthographic markers of word boundaries. A “greedy” approach is to scan the
 1913 input for character substrings that are in a predefined lexicon. However, Xue et al. (2003)
 1914 notes that this can be ambiguous, since many character sequences could be segmented in
 1915 multiple ways. Instead, he trains a classifier to determine whether each Chinese character,
 1916 or **hanzi**, is a word boundary. More advanced sequence labeling methods for word seg-
 1917 mentation are discussed in § 8.4. Similar problems can occur in languages with alphabetic
 1918 scripts, such as German, which does not include whitespace in compound nouns, yield-
 1919 ing examples such as *Freundschaftsbezeugungen* (demonstration of friendship) and *Dilett-*
 1920 *tantenaufdringlichkeiten* (the importunities of dilettantes). As Twain (1997) argues, “*These*
 1921 *things are not words, they are alphabetic processions.*” Social media raises similar problems
 1922 for English and other languages, with hashtags such as `#TrueLoveInFourWords` requiring
 1923 decomposition for analysis (Brun and Roux, 2014).

1924 4.3.1.2 Normalization

1925 After splitting the text into tokens, the next question is which tokens are really distinct.
 1926 Is it necessary to distinguish *great*, *Great*, and *GREAT*? Sentence-initial capitalization may
 1927 be irrelevant to the classification task. Going further, the complete elimination of case
 1928 distinctions will result in a smaller vocabulary, and thus smaller feature vectors. However,

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

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

1929 case distinctions might be relevant in some situations: for example, *apple* is a delicious
 1930 pie filling, while *Apple* is a company that specializes in proprietary dongles and power
 1931 adapters.

1932 For Roman script, case conversion can be performed using unicode string libraries.
 1933 Many scripts do not have case distinctions (e.g., the Devanagari script used for South
 1934 Asian languages, the Thai alphabet, and Japanese kana), and case conversion for all scripts
 1935 may not be available in every programming environment. (Unicode support is an im-
 1936 portant distinction between Python’s versions 2 and 3, and is a good reason for mi-
 1937 grating to Python 3 if you have not already done so. Compare the output of the code
 1938 "\à l\'hôtel".upper() in the two language versions.)⁶

1939 Case conversion is a type of **normalization**, which refers to string transformations that
 1940 remove distinctions that are irrelevant to downstream applications (Sproat et al., 2001).
 1941 Other normalizations include the standardization of numbers (e.g., 1,000 to 1000) and
 1942 dates (e.g., August 11, 2015 to 2015/11/08). Depending on the application, it may even be
 1943 worthwhile to convert all numbers and dates to special tokens, !NUM and !DATE. In social
 1944 media, there are additional orthographic phenomena that may be normalized, such as ex-
 1945 pressive lengthening, e.g., *coooooool* (Aw et al., 2006; Yang and Eisenstein, 2013). Similarly,
 1946 historical texts feature spelling variations that may need to be normalized to a contempo-
 1947 rary standard form (Baron and Rayson, 2008).

1948 A more extreme form of normalization is to eliminate **inflectional affixes**, such as the
 1949 *-ed* and *-s* suffixes in English. On this view, *bike*, *bikes*, *biking*, and *biked* all refer to the
 1950 same underlying concept, so they should be grouped into a single feature. A **stemmer** is
 1951 a program for eliminating affixes, usually by applying a series of regular expression sub-
 1952 stitutions. Character-based stemming algorithms are necessarily approximate, as shown
 1953 in Figure 4.2: the Lancaster stemmer incorrectly identifies *-ers* as an inflectional suffix of
 1954 *sisters* (by analogy to *fix/fixer*), and both stemmers incorrectly identify *-s* as a suffix of *this*
 1955 and *Williams*. Fortunately, even inaccurate stemming can improve bag-of-words classifi-
 1956 cation models, by merging related strings and thereby reducing the vocabulary size.

1957 Accurately handling irregular orthography requires word-specific rules. **Lemmatizers**

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

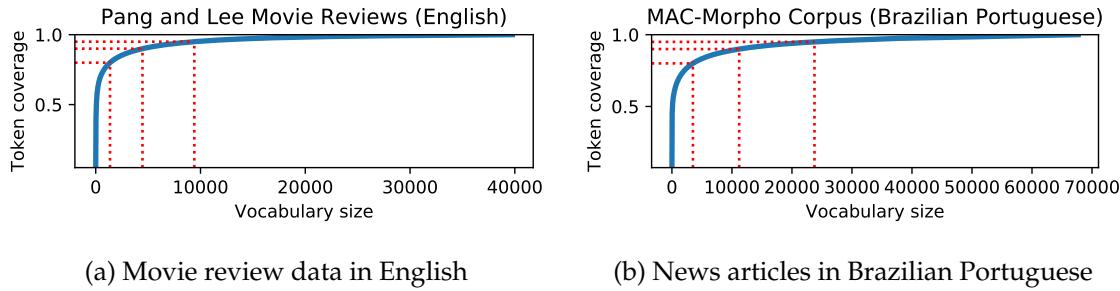


Figure 4.3: Tradeoff between token coverage (y-axis) and vocabulary size, on the `nltk` movie review dataset, after sorting the vocabulary by decreasing frequency. The red dashed lines indicate 80%, 90%, and 95% coverage.

1958 are systems that identify the underlying lemma of a given wordform. They must avoid the
 1959 over-generalization errors of the stemmers in Figure 4.2, and also handle more complex
 1960 transformations, such as *geese*→*goose*. The output of the WordNet lemmatizer is shown in
 1961 the final line of Figure 4.2. Both stemming and lemmatization are language-specific: an
 1962 English stemmer or lemmatizer is of little use on a text written in another language. The
 1963 discipline of **morphology** relates to the study of word-internal structure, and is described
 1964 in more detail in § 9.1.2.

1965 The value of normalization depends on the data and the task. Normalization re-
 1966 duces the size of the feature space, which can help in generalization. However, there
 1967 is always the risk of merging away linguistically meaningful distinctions. In supervised
 1968 machine learning, regularization and smoothing can play a similar role to normalization
 1969 — preventing the learner from overfitting to rare features — while avoiding the language-
 1970 specific engineering required for accurate normalization. In unsupervised scenarios, such
 1971 as content-based information retrieval (Manning et al., 2008) and topic modeling (Blei
 1972 et al., 2003), normalization is more critical.

1973 4.3.2 How many words?

1974 Limiting the size of the feature vector reduces the memory footprint of the resulting mod-
 1975 els, and increases the speed of prediction. Normalization can help to play this role, but
 1976 a more direct approach is simply to limit the vocabulary to the N most frequent words
 1977 in the dataset. For example, in the movie-reviews dataset provided with `nltk` (orig-
 1978 inally from Pang et al., 2002), there are 39,768 word types, and 1.58M tokens. As shown
 1979 in Figure 4.3a, the most frequent 4000 word types cover 90% of all tokens, offering an
 1980 order-of-magnitude reduction in the model size. Such ratios are language-specific: in for
 1981 example, in the Brazilian Portuguese Mac-Morpho corpus (Aluísio et al., 2003), attain-
 1982 ing 90% coverage requires more than 10000 word types (Figure 4.3b). This reflects the

1983 morphological complexity of Portuguese, which includes many more inflectional suffixes
 1984 than English.

1985 Eliminating rare words is not always advantageous for classification performance: for
 1986 example, names, which are typically rare, play a large role in distinguishing topics of news
 1987 articles. Another way to reduce the size of the feature space is to eliminate **stopwords** such
 1988 as *the*, *to*, and *and*, which may seem to play little role in expressing the topic, sentiment,
 1989 or stance. This is typically done by creating a **stoplist** (e.g., `nltk.corpus.stopwords`),
 1990 and then ignoring all terms that match the list. However, corpus linguists and social psy-
 1991 chologists have shown that seemingly inconsequential words can offer surprising insights
 1992 about the author or nature of the text (Biber, 1991; Chung and Pennebaker, 2007). Further-
 1993 more, high-frequency words are unlikely to cause overfitting in discriminative classifiers.
 1994 As with normalization, stopword filtering is more important for unsupervised problems,
 1995 such as term-based document retrieval.

1996 Another alternative for controlling model size is **feature hashing** (Weinberger et al.,
 1997 2009). Each feature is assigned an index using a hash function. If a hash function that
 1998 permits collisions is chosen (typically by taking the hash output modulo some integer),
 1999 then the model can be made arbitrarily small, as multiple features share a single weight.
 2000 Because most features are rare, accuracy is surprisingly robust to such collisions (Ganchev
 2001 and Dredze, 2008).

2002 4.3.3 Count or binary?

2003 Finally, we may consider whether we want our feature vector to include the **count** of each
 2004 word, or its **presence**. This gets at a subtle limitation of linear classification: it worse to
 2005 have two *failures* than one, but is it really twice as bad? Motivated by this intuition, Pang
 2006 et al. (2002) use binary indicators of presence or absence in the feature vector: $f_j(x, y) \in$
 2007 $\{0, 1\}$. They find that classifiers trained on these binary vectors tend to outperform feature
 2008 vectors based on word counts. One explanation is that words tend to appear in clumps:
 2009 if a word has appeared once in a document, it is likely to appear again (Church, 2000).
 2010 These subsequent appearances can be attributed to this tendency towards repetition, and
 2011 thus provide little additional information about the class label of the document.

2012 4.4 Evaluating classifiers

2013 In any supervised machine learning application, it is critical to reserve a held-out test set.
 2014 This data should be used for only one purpose: to evaluate the overall accuracy of a single
 2015 classifier. Using this data more than once would cause the estimated accuracy to be overly
 2016 optimistic, because the classifier would be customized to this data, and would not perform
 2017 as well as on unseen data in the future. It is usually necessary to set hyperparameters or

2018 perform feature selection, so you may need to construct a **tuning** or **development set** for
 2019 this purpose, as discussed in § 2.1.5.

2020 There are a number of ways to evaluate classifier performance. The simplest is **accuracy**:
 2021 the number of correct predictions, divided by the total number of instances,

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

2022 Exams are usually graded by accuracy. Why are other metrics necessary? The main
 2023 reason is **class imbalance**. Suppose you are building a classifier to detect whether an
 2024 electronic health record (EHR) describes symptoms of a rare disease, which appears in
 2025 only 1% of all documents in the dataset. A classifier that reports $\hat{y} = \text{NEGATIVE}$ for
 2026 all documents would achieve 99% accuracy, but would be practically useless. We need
 2027 metrics that are capable of detecting the classifier's ability to discriminate between classes,
 2028 even when the distribution is skewed.

2029 One solution is to build a **balanced test set**, in which each possible label is equally rep-
 2030 resented. But in the EHR example, this would mean throwing away 98% of the original
 2031 dataset! Furthermore, the detection threshold itself might be a design consideration: in
 2032 health-related applications, we might prefer a very sensitive classifier, which returned a
 2033 positive prediction if there is even a small chance that $y^{(i)} = \text{POSITIVE}$. In other applica-
 2034 tions, a positive result might trigger a costly action, so we would prefer a classifier that
 2035 only makes positive predictions when absolutely certain. We need additional metrics to
 2036 capture these characteristics.

2037 4.4.1 Precision, recall, and F-MEASURE

2038 For any label (e.g., positive for presence of symptoms of a disease), there are two possible
 2039 errors:

- 2040 • **False positive**: the system incorrectly predicts the label.
- 2041 • **False negative**: the system incorrectly fails to predict the label.

2042 Similarly, for any label, there are two ways to be correct:

- 2043 • **True positive**: the system correctly predicts the label.
- 2044 • **True negative**: the system correctly predicts that the label does not apply to this
 2045 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**:

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

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

2046 Recall and precision are both conditional likelihoods of a correct prediction, which is why
 2047 their numerators are the same. Recall is conditioned on k being the correct label, $y^{(i)} = k$,
 2048 so the denominator sums over true positive and false negatives. Precision is conditioned
 2049 on k being the prediction, so the denominator sums over true positives and false positives.
 2050 Note that true negatives are not considered in either statistic. The classifier that labels
 2051 every document as “negative” would achieve zero recall; precision would be $\frac{0}{0}$.

2052 Recall and precision are complementary. A high-recall classifier is preferred when
 2053 false negatives are cheaper than false positives: for example, in a preliminary screening
 2054 for symptoms of a disease, the cost of a false positive might be an additional test, while a
 2055 false negative would result in the disease going untreated. Conversely, a high-precision
 2056 classifier is preferred when false positives are more expensive: for example, in spam de-
 2057tection, a false negative is a relatively minor inconvenience, while a false positive might
 2058 mean that an important message goes unread.

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

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

2059 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,

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

2060 In multi-class problems with unbalanced class distributions, the macro ***F*-MEASURE** is a
 2061 balanced measure of how well the classifier recognizes each class. In **micro *F*-MEASURE**,
 2062 we compute true positives, false positives, and false negatives for each class, and then add
 2063 them up to compute a single recall, precision, and ***F*-MEASURE**. This metric is balanced
 2064 across instances rather than classes, so it weights each class in proportion to its frequency
 2065 — unlike macro ***F*-MEASURE**, which weights each class equally.

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

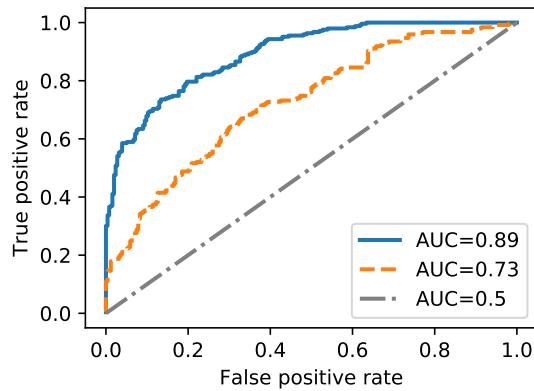


Figure 4.4: ROC curves for three classifiers of varying discriminative power, measured by AUC (area under the curve)

2066 4.4.2 Threshold-free metrics

2067 In binary classification problems, it is possible to trade off between recall and precision by
 2068 adding a constant “threshold” to the output of the scoring function. This makes it possible
 2069 to trace out a curve, where each point indicates the performance at a single threshold. In
 2070 the **receiver operating characteristic (ROC)** curve,⁸ the *x*-axis indicates the **false positive**
 2071 **rate**, $\frac{FP}{FP+TN}$, and the *y*-axis indicates the recall, or **true positive rate**. A perfect classifier
 2072 attains perfect recall without any false positives, tracing a “curve” from the origin (0,0) to
 2073 the upper left corner (0,1), and then to (1,1). In expectation, a non-discriminative classifier
 2074 traces a diagonal line from the origin (0,0) to the upper right corner (1,1). Real classifiers
 2075 tend to fall between these two extremes. Examples are shown in Figure 4.4.

2076 The ROC curve can be summarized in a single number by taking its integral, the **area**
 2077 **under the curve (AUC)**. The AUC can be interpreted as the probability that a randomly-
 2078 selected positive example will be assigned a higher score by the classifier than a randomly-
 2079 selected negative example. A perfect classifier has AUC = 1 (all positive examples score
 2080 higher than all negative examples); a non-discriminative classifier has AUC = 0.5 (given
 2081 a randomly selected positive and negative example, either could score higher with equal
 2082 probability); a perfectly wrong classifier would have AUC = 0 (all negative examples score
 2083 higher than all positive examples). One advantage of AUC in comparison to *F*-MEASURE
 2084 is that the baseline rate of 0.5 does not depend on the label distribution.

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

2085 **4.4.3 Classifier comparison and statistical significance**

2086 Natural language processing research and engineering often involves comparing different
 2087 classification techniques. In some cases, the comparison is between algorithms, such as
 2088 logistic regression versus averaged perceptron, or L_2 regularization versus L_1 . In other
 2089 cases, the comparison is between feature sets, such as the bag-of-words versus positional
 2090 bag-of-words (see § 4.2.2). **Ablation testing** involves systematically removing (ablating)
 2091 various aspects of the classifier, such as feature groups, and testing the **null hypothesis**
 2092 that the ablated classifier is as good as the full model.

2093 A full treatment of hypothesis testing is beyond the scope of this text, but this section
 2094 contains a brief summary of the techniques necessary to compare classifiers. The main
 2095 aim of hypothesis testing is to determine whether the difference between two statistics
 2096 — for example, the accuracies of two classifiers — is likely to arise by chance. We will
 2097 be concerned with chance fluctuations that arise due to the finite size of the test set.⁹ An
 2098 improvement of 10% on a test set with ten instances may reflect a random fluctuation that
 2099 makes the test set more favorable to classifier c_1 than c_2 ; on another test set with a different
 2100 ten instances, we might find that c_2 does better than c_1 . But if we observe the same 10%
 2101 improvement on a test set with 1000 instances, this is highly unlikely to be explained
 2102 by chance. Such a finding is said to be **statistically significant** at a level p , which is the
 2103 probability of observing an effect of equal or greater magnitude when the null hypothesis
 2104 is true. The notation $p < .05$ indicates that the likelihood of an equal or greater effect is
 2105 less than 5%, assuming the null hypothesis is true.¹⁰

2106 **4.4.3.1 The binomial test**

2107 The statistical significance of a difference in accuracy can be evaluated using classical tests,
 2108 such as the **binomial test**.¹¹ Suppose that classifiers c_1 and c_2 disagree on N instances in a
 2109 test set with binary labels, and that c_1 is correct on k of those instances. Under the null hy-
 2110 pothesis that the classifiers are equally accurate, we would expect k/N to be roughly equal
 2111 to 1/2, and as N increases, k/N should be increasingly close to this expected value. These
 2112 properties are captured by the **binomial distribution**, which is a probability over counts

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

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

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

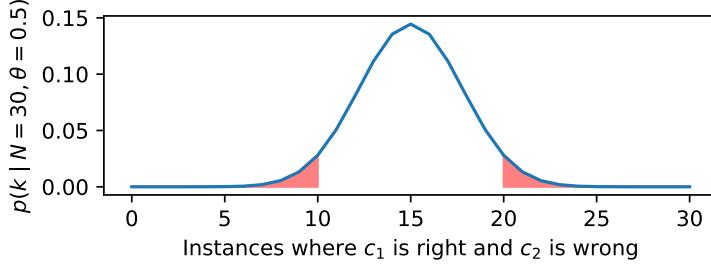


Figure 4.5: Probability mass function for the binomial distribution. The pink highlighted areas represent the cumulative probability for a significance test on an observation of $k = 10$ and $N = 30$.

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

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

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

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

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

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

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

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

```

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

2127 rate, we would take the sum of one-tailed p -values, where the second term is computed
 2128 from the right tail of Figure 4.5. The binomial distribution is symmetric, so this can be
 2129 computed by simply doubling the one-tailed p -value.

2130 Two-tailed tests are more stringent, but they are necessary in cases in which there is
 2131 no prior intuition about whether c_1 or c_2 is better. For example, in comparing logistic
 2132 regression versus averaged perceptron, a two-tailed test is appropriate. In an ablation
 2133 test, c_2 may contain a superset of the features available to c_1 . If the additional features are
 2134 thought to be likely to improve performance, then a one-tailed test would be appropriate,
 2135 if chosen in advance. However, such a test can only prove that c_2 is more accurate than
 2136 c_1 , and not the reverse.

2137 **4.4.3.2 *Randomized testing**

2138 The binomial test is appropriate for accuracy, but not for more complex metrics such as
 2139 F -MEASURE. To compute statistical significance for arbitrary metrics, we can apply ran-
 2140 domization. Specifically, draw a set of M **bootstrap samples** (Efron and Tibshirani, 1993),
 2141 by resampling instances from the original test set with replacement. Each bootstrap sam-
 2142 ple is itself a test set of size N . Some instances from the original test set will not appear
 2143 in any given bootstrap sample, while others will appear multiple times; but overall, the
 2144 sample will be drawn from the same distribution as the original test set. We can then com-
 2145 pute any desired evaluation on each bootstrap sample, which gives a distribution over the
 2146 value of the metric. Algorithm 7 shows how to perform this computation.

2147 To compare the F -MEASURE of two classifiers c_1 and c_2 , we set the function $\delta(\cdot)$ to
 2148 compute the difference in F -MEASURE on the bootstrap sample. If the difference is less
 2149 than or equal to zero in at least 5% of the samples, then we cannot reject the one-tailed
 2150 null hypothesis that c_2 is at least as good as c_1 (Berg-Kirkpatrick et al., 2012). We may
 2151 also be interested in the 95% **confidence interval** around a metric of interest, such as
 2152 the F -MEASURE of a single classifier. This can be computed by sorting the output of

2153 Algorithm 7, and then setting the top and bottom of the 95% confidence interval to the
 2154 values at the 2.5% and 97.5% percentiles of the sorted outputs. Alternatively, you can fit
 2155 a normal distribution to the set of differences across bootstrap samples, and compute a
 2156 Gaussian confidence interval from the mean and variance.

2157 As the number of bootstrap samples goes to infinity, $M \rightarrow \infty$, the bootstrap estimate
 2158 is increasingly accurate. A typical choice for M is 10^4 or 10^5 ; larger numbers of samples
 2159 are necessary for smaller p -values. One way to validate your choice of M is to run the test
 2160 multiple times, and ensure that the p -values are similar; if not, increase M by an order of
 2161 magnitude. This is a heuristic measure of the **variance** of the test, which can decrease
 2162 with the square root \sqrt{M} (Robert and Casella, 2013).

2163 4.4.4 *Multiple comparisons

2164 Sometimes it is necessary to perform multiple hypothesis tests, such as when comparing
 2165 the performance of several classifiers on multiple datasets. Suppose you have five
 2166 datasets, and you compare four versions of your classifier against a baseline system, for a
 2167 total of 20 comparisons. Even if none of your classifiers is better than the baseline, there
 2168 will be some chance variation in the results, and in expectation you will get one statisti-
 2169 cally significant improvement at $p = 0.05 = \frac{1}{20}$. It is therefore necessary to adjust the
 2170 p -values when reporting the results of multiple comparisons.

2171 One approach is to require a threshold of $\frac{\alpha}{m}$ to report a p value of $p < \alpha$ when per-
 2172 forming m tests. This is known as the **Bonferroni correction**, and it limits the overall
 2173 probability of incorrectly rejecting the null hypothesis at α . Another approach is to bound
 2174 the **false discovery rate** (FDR), which is the fraction of null hypothesis rejections that are
 2175 incorrect. Benjamini and Hochberg (1995) propose a p -value correction that bounds the
 2176 fraction of false discoveries at α : sort the p -values of each individual test in ascending
 2177 order, and set the significance threshold equal to largest k such that $p_k \leq \frac{k}{m}\alpha$. If $k > 1$, the
 2178 FDR adjustment is more permissive than the Bonferroni correction.

2179 4.5 Building datasets

2180 Sometimes, if you want to build a classifier, you must first build a dataset of your own.
 2181 This includes selecting a set of documents or instances to annotate, and then performing
 2182 the annotations. The scope of the dataset may be determined by the application: if you
 2183 want to build a system to classify electronic health records, then you must work with a
 2184 corpus of records of the type that your classifier will encounter when deployed. In other
 2185 cases, the goal is to build a system that will work across a broad range of documents. In
 2186 this case, it is best to have a *balanced* corpus, with contributions from many styles and
 2187 genres. For example, the Brown corpus draws from texts ranging from government doc-
 2188 uments to romance novels (Francis, 1964), and the Google Web Treebank includes an-

2189 notations for five “domains” of web documents: question answers, emails, newsgroups,
2190 reviews, and blogs (Petrov and McDonald, 2012).

2191 4.5.1 Metadata as labels

2192 Annotation is difficult and time-consuming, and most people would rather avoid it. It
2193 is sometimes possible to exploit existing metadata to obtain labels for training a classi-
2194 fier. For example, reviews are often accompanied by a numerical rating, which can be
2195 converted into a classification label (see § 4.1). Similarly, the nationalities of social media
2196 users can be estimated from their profiles (Dredze et al., 2013) or even the time zones of
2197 their posts (Gouws et al., 2011). More ambitiously, we may try to classify the political af-
2198 filiations of social media profiles based on their social network connections to politicians
2199 and major political parties (Rao et al., 2010).

2200 The convenience of quickly constructing large labeled datasets without manual an-
2201 notation is appealing. However this approach relies on the assumption that unlabeled
2202 instances — for which metadata is unavailable — will be similar to labeled instances.
2203 Consider the example of labeling the political affiliation of social media users based on
2204 their network ties to politicians. If a classifier attains high accuracy on such a test set,
2205 is it safe to assume that it accurately predicts the political affiliation of all social media
2206 users? Probably not. Social media users who establish social network ties to politicians
2207 may be more likely to mention politics in the text of their messages, as compared to the
2208 average user, for whom no political metadata is available. If so, the accuracy on a test set
2209 constructed from social network metadata would give an overly optimistic picture of the
2210 method’s true performance on unlabeled data.

2211 4.5.2 Labeling data

2212 In many cases, there is no way to get ground truth labels other than manual annotation.
2213 An annotation protocol should satisfy several criteria: the annotations should be *expressive*
2214 enough to capture the phenomenon of interest; they should be *replicable*, meaning that
2215 another annotator or team of annotators would produce very similar annotations if given
2216 the same data; and they should be *scalable*, so that they can be produced relatively quickly.
2217 Hovy and Lavid (2010) propose a structured procedure for obtaining annotations that
2218 meet these criteria, which is summarized below.

- 2219 1. **Determine what the annotations are to include.** This is usually based on some
2220 theory of the underlying phenomenon: for example, if the goal is to produce an-
2221 notations about the emotional state of a document’s author, one should start with a
2222 theoretical account of the types or dimensions of emotion (e.g., Mohammad and Tur-
2223 ney, 2013). At this stage, the tradeoff between expressiveness and scalability should

2224 be considered: a full instantiation of the underlying theory might be too costly to
2225 annotate at scale, so reasonable approximations should be considered.

- 2226 2. Optionally, one may **design or select a software tool to support the annotation**
2227 **effort**. Existing general-purpose annotation tools include BRAT (Stenetorp et al.,
2228 2012) and MMAX2 (Müller and Strube, 2006).
- 2229 3. **Formalize the instructions for the annotation task.** To the extent that the instruc-
2230 tions are not explicit, the resulting annotations will depend on the intuitions of the
2231 annotators. These intuitions may not be shared by other annotators, or by the users
2232 of the annotated data. Therefore explicit instructions are critical to ensuring the an-
2233 notations are replicable and usable by other researchers.
- 2234 4. **Perform a pilot annotation** of a small subset of data, with multiple annotators for
2235 each instance. This will give a preliminary assessment of both the replicability and
2236 scalability of the current annotation instructions. Metrics for computing the rate of
2237 agreement are described below. Manual analysis of specific disagreements should
2238 help to clarify the instructions, and may lead to modifications of the annotation task
2239 itself. For example, if two labels are commonly conflated by annotators, it may be
2240 best to merge them.
- 2241 5. **Annotate the data.** After finalizing the annotation protocol and instructions, the
2242 main annotation effort can begin. Some, if not all, of the instances should receive
2243 multiple annotations, so that inter-annotator agreement can be computed. In some
2244 annotation projects, instances receive many annotations, which are then aggregated
2245 into a “consensus” label (e.g., Danescu-Niculescu-Mizil et al., 2013). However, if the
2246 annotations are time-consuming or require significant expertise, it may be preferable
2247 to maximize scalability by obtaining multiple annotations for only a small subset of
2248 examples.
- 2249 6. **Compute and report inter-annotator agreement, and release the data.** In some
2250 cases, the raw text data cannot be released, due to concerns related to copyright or
2251 privacy. In these cases, one solution is to publicly release **stand-off annotations**,
2252 which contain links to document identifiers. The documents themselves can be re-
2253 leased under the terms of a licensing agreement, which can impose conditions on
2254 how the data is used. It is important to think through the potential consequences of
2255 releasing data: people may make personal data publicly available without realizing
2256 that it could be redistributed in a dataset and publicized far beyond their expecta-
2257 tions (boyd and Crawford, 2012).

2258 **4.5.2.1 Measuring inter-annotator agreement**

2259 To measure the replicability of annotations, a standard practice is to compute the extent to
 2260 which annotators agree with each other. If the annotators frequently disagree, this casts
 2261 doubt on either their reliability or on the annotation system itself. For classification, one
 2262 can compute the frequency with which the annotators agree; for rating scales, one can
 2263 compute the average distance between ratings. These raw agreement statistics must then
 2264 be compared with the rate of **chance agreement** — the level of agreement that would be
 2265 obtained between two annotators who ignored the data.

2266 **Cohen's Kappa** is widely used for quantifying the agreement on discrete labeling
 2267 tasks (Cohen, 1960; Carletta, 1996),¹²

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

2268 The numerator is the difference between the observed agreement and the chance agree-
 2269 ment, and the denominator is the difference between perfect agreement and chance agree-
 2270 ment. Thus, $\kappa = 1$ when the annotators agree in every case, and $\kappa = 0$ when the annota-
 2271 tors agree only as often as would happen by chance. Various heuristic scales have been
 2272 proposed for determining when κ indicates "moderate", "good", or "substantial" agree-
 2273 ment; for reference, Lee and Narayanan (2005) report $\kappa \approx 0.45 - 0.47$ for annotations
 2274 of emotions in spoken dialogues, which they describe as "moderate agreement"; Stolcke
 2275 et al. (2000) report $\kappa = 0.8$ for annotations of **dialogue acts**, which are labels for the pur-
 2276 pose of each turn in a conversation.

2277 When there are two annotators, the expected chance agreement is computed as,

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

2278 where k is a sum over labels, and $\hat{\Pr}(Y = k)$ is the empirical probability of label k across
 2279 all annotations. The formula is derived from the expected number of agreements if the
 2280 annotations were randomly shuffled. Thus, in a binary labeling task, if one label is applied
 2281 to 90% of instances, chance agreement is $.9^2 + .1^2 = .82$.

2282 **4.5.2.2 Crowdsourcing**

2283 Crowdsourcing is often used to rapidly obtain annotations for classification problems.
 2284 For example, **Amazon Mechanical Turk** makes it possible to define "human intelligence
 2285 tasks (hits)", such as labeling data. The researcher sets a price for each set of annotations
 2286 and a list of minimal qualifications for annotators, such as their native language and their

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

2287 satisfaction rate on previous tasks. The use of relatively untrained “crowdworkers” con-
 2288 trasts with earlier annotation efforts, which relied on professional linguists (Marcus et al.,
 2289 1993). However, crowdsourcing has been found to produce reliable annotations for many
 2290 language-related tasks (Snow et al., 2008). Crowdsourcing is part of the broader field of
 2291 **human computation** (Law and Ahn, 2011).

2292 Additional resources

2293 Many of the preprocessing issues discussed in this chapter also arise in information re-
 2294 trieval. See (Manning et al., 2008, chapter 2) for discussion of tokenization and related
 2295 algorithms.

2296 Exercises

2297 1. As noted in § 4.3.3, words tend to appear in clumps, with subsequent occurrences
 2298 of a word being more probable. More concretely, if word j has probability $\phi_{y,j}$
 2299 of appearing in a document with label y , then the probability of two appearances
 2300 ($x_j^{(i)} = 2$) is greater than $\phi_{y,j}^2$.

2301 Suppose you are applying Naïve Bayes to a binary classification. Focus on a word j
 2302 which is more probable under label $y = 1$, so that,

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

2303 Now suppose that $x_j^{(i)} > 1$. All else equal, will the classifier overestimate or under-
 2304 estimate the posterior $\Pr(y = 1 \mid x)$?

2305 2. Prove that F-measure is never greater than the arithmetic mean of recall and pre-
 2306 cision, $\frac{r+p}{2}$. Your solution should also show that F-measure is equal to $\frac{r+p}{2}$ iff $r = p$.

2307 3. Given a binary classification problem in which the probability of the “positive” label
 2308 is equal to α , what is the expected *F*-MEASURE of a random classifier which ignores
 2309 the data, and selects $\hat{y} = +1$ with probability $\frac{1}{2}$? (Assume that $p(\hat{y}) \perp p(y)$.) What is
 2310 the expected *F*-MEASURE of a classifier that selects $\hat{y} = +1$ with probability α (also
 2311 independent of $y^{(i)}$)? Depending on α , which random classifier will score better?

2312 4. Suppose that binary classifiers c_1 and c_2 disagree on $N = 30$ cases, and that c_1 is
 2313 correct in $k = 10$ of those cases.

- 2314 • Write a program that uses primitive functions such as `exp` and `factorial` to com-
 2315 pute the **two-tailed** *p*-value — you may use an implementation of the “choose”
 2316 function if one is available. Verify your code against the output of a library for

- 2317 computing the binomial test or the binomial CDF, such as `scipy.stats.binom`
 2318 in Python.
- 2319 • Then use a randomized test to try to obtain the same p -value. In each sample,
 2320 draw from a binomial distribution with $N = 30$ and $\theta = \frac{1}{2}$. Count the fraction
 2321 of samples in which $k \leq 10$. This is the one-tailed p -value; double this to
 2322 compute the two-tailed p -value.
 - 2323 • Try this with varying numbers of bootstrap samples: $M \in \{100, 1000, 5000, 10000\}$.
 2324 For $M = 100$ and $M = 1000$, run the test 10 times, and plot the resulting p -
 2325 values.
 - 2326 • Finally, perform the same tests for $N = 70$ and $k = 25$.
- 2327 5. SemCor 3.0 is a labeled dataset for word sense disambiguation. You can download
 2328 it,¹³ or access it in `nltk.corpora.semcor`.
- 2329 Choose a word that appears at least ten times in SemCor (*find*), and annotate its
 2330 WordNet senses across ten randomly-selected examples, without looking at the ground
 2331 truth. Use online WordNet to understand the definition of each of the senses.¹⁴ Have
 2332 a partner do the same annotations, and compute the raw rate of agreement, expected
 2333 chance rate of agreement, and Cohen's kappa.
- 2344 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
 2345 randomly-selected 400 reviews as a test set.
- 2346 Download a sentiment lexicon, such as the one currently available from Bing Liu,
 2347 <https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html>. Tokenize
 2348 the data, and classify each document as positive iff it has more positive sentiment
 2349 words than negative sentiment words. Compute the accuracy and *F*-MEASURE on
 2350 detecting positive reviews on the test set, using this lexicon-based classifier.
- 2351 Then train a discriminative classifier (averaged perceptron or logistic regression) on
 2352 the training set, and compute its accuracy and *F*-MEASURE on the test set.
- 2353 Determine whether the differences are statistically significant, using two-tailed hy-
 2354 pothesis tests: Binomial for the difference in accuracy, and bootstrap for the differ-
 2355 ence in macro-*F*-MEASURE.
- 2356 2347 The remaining problems will require you to build a classifier and test its properties. Pick
 2348 a multi-class text classification dataset, such as RCV1¹⁵). Divide your data into training

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

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

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

2349 (60%), development (20%), and test sets (20%), if no such division already exists. [todo:
2350 this dataset is already tokenized, find something else]

2351 7. Compare various vocabulary sizes of $10^2, 10^3, 10^4, 10^5$, using the most frequent words
2352 in each case (you may use any reasonable tokenizer). Train logistic regression clas-
2353 sifiers for each vocabulary size, and apply them to the development set. Plot the
2354 accuracy and Macro-*F*-MEASURE with the increasing vocabulary size. For each vo-
2355 cabulary size, tune the regularizer to maximize accuracy on a subset of data that is
2356 held out from the training set.

2357 8. Compare the following tokenization algorithms:

- 2358 • Whitespace, using a regular expression
2359 • Penn Treebank
2360 • Split input into five-character units, regardless of whitespace or punctuation

2361 Compute the token/type ratio for each tokenizer on the training data, and explain
2362 what you find. Train your classifier on each tokenized dataset, tuning the regularizer
2363 on a subset of data that is held out from the training data. Tokenize the development
2364 set, and report accuracy and Macro-*F*-MEASURE.

2365 9. Apply the Porter and Lancaster stemmers to the training set, using any reasonable
2366 tokenizer, and compute the token/type ratios. Train your classifier on the stemmed
2367 data, and compute the accuracy and Macro-*F*-MEASURE on stemmed development
2368 data, again using a held-out portion of the training data to tune the regularizer.

2369 10. Identify the best combination of vocabulary filtering, tokenization, and stemming
2370 from the previous three problems. Apply this preprocessing to the test set, and
2371 compute the test set accuracy and Macro-*F*-MEASURE. Compare against a baseline
2372 system that applies no vocabulary filtering, whitespace tokenization, and no stem-
2373 ming.

2374 Use the binomial test to determine whether your best-performing system is signifi-
2375 cantly more accurate than the baseline.

2376 Use the bootstrap test with $M = 10^4$ to determine whether your best-performing
2377 system achieves significantly higher macro-*F*-MEASURE.

2378 Chapter 5

2379 Learning without supervision

2380 So far we've assumed the following setup:

- 2381 a **training set** where you get observations x and labels y ;
- 2382 a **test set** where you only get observations x .

2383 Without labeled data, is it possible to learn anything? This scenario is known as **unsu-**
2384 **pervised learning**, and we will see that indeed it is possible to learn about the underlying
2385 structure of unlabeled observations. This chapter will also explore some related scenarios:
2386 **semi-supervised learning**, in which only some instances are labeled, and **domain adap-**
2387 **tation**, in which the training data differs from the data on which the trained system will
2388 be deployed.

2389 5.1 Unsupervised learning

2390 To motivate unsupervised learning, consider the problem of word sense disambiguation
2391 (§ 4.2). Our goal is to classify each instance of a word, such as *bank* into a sense,

- 2392 bank#1: a financial institution
- 2393 bank#2: the land bordering a river

2394 It is difficult to obtain sufficient training data for word sense disambiguation, because
2395 even a large corpus will contain only a few instances of all but the most common words.
2396 Is it possible to learn anything about these different senses without labeled data?

2397 Word sense disambiguation is usually performed using feature vectors constructed
2398 from the local context of the word to be disambiguated. For example, for the word

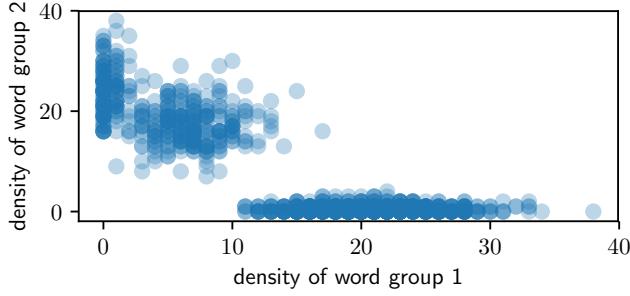


Figure 5.1: Counts of words from two different context groups

2399 *bank*, the immediate context might typically include words from one of the following two
 2400 groups:

- 2401 1. *financial, deposits, credit, lending, capital, markets, regulated, reserve, liquid, assets*
 2402 2. *land, water, geography, stream, river, flow, deposits, discharge, channel, ecology*

2403 Now consider a scatterplot, in which each point is a document containing the word *bank*.
 2404 The location of the document on the x -axis is the count of words in group 1, and the
 2405 location on the y -axis is the count for group 2. In such a plot, shown in Figure 5.1, two
 2406 “blobs” might emerge, and these blobs correspond to the different senses of *bank*.

2407 Here’s a related scenario, from a different problem. Suppose you download thousands
 2408 of news articles, and make a scatterplot, where each point corresponds to a document:
 2409 the x -axis is the frequency of the group of words (*hurricane, winds, storm*); the y -axis is the
 2410 frequency of the group (*election, voters, vote*). This time, three blobs might emerge: one
 2411 for documents that are largely about a hurricane, another for documents largely about a
 2412 election, and a third for documents about neither topic.

2413 These clumps represent the underlying structure of the data. But the two-dimensional
 2414 scatter plots are based on groupings of context words, and in real scenarios these word
 2415 lists are unknown. Unsupervised learning applies the same basic idea, but in a high-
 2416 dimensional space with one dimension for every context word. This space can’t be di-
 2417 rectly visualized, but the idea is the same: try to identify the underlying structure of the
 2418 observed data, such that there are a few clusters of points, each of which is internally
 2419 coherent. **Clustering** algorithms are capable of finding such structure automatically.

2420 5.1.1 **K-means** clustering

2421 Clustering algorithms assign each data point to a discrete cluster, $z_i \in 1, 2, \dots, K$. One of
 2422 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

```

2423 a cluster assignment for each instance, and a central (“mean”) location for each cluster.
 2424 K -means iterates between updates to the assignments and the centers:

2425 1. each instance is placed in the cluster with the closest center;

2426 2. each center is recomputed as the average over points in the cluster.

2427 This is formalized in Algorithm 8. The term $\|\mathbf{x}^{(i)} - \boldsymbol{\nu}\|^2$ refers to the squared Euclidean
 2428 norm, $\sum_{j=1}^V (x_j^{(i)} - \nu_j)^2$.

2429 **Soft K -means** is a particularly relevant variant. Instead of directly assigning each
 2430 point to a specific cluster, soft K -means assigns each point a **distribution** over clusters
 2431 $\mathbf{q}^{(i)}$, 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
 2432 the distance of $\mathbf{x}^{(i)}$ to the cluster center $\boldsymbol{\nu}_k$. In turn, the center of each cluster is computed
 2433 from a **weighted average** of the points in the cluster,

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

2434 We will now explore a probabilistic version of soft K -means clustering, based on **expectation**
 2435 **maximization** (EM). Because EM clustering can be derived as an approximation to
 2436 maximum-likelihood estimation, it can be extended in a number of useful ways.

2437 5.1.2 Expectation Maximization (EM)

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

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

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

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

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

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

2438 To estimate the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$, we can maximize the marginal likelihood in Equation 5.5. Why is this the right thing to maximize? Without labels, discriminative learning 2439 is impossible — there's nothing to discriminate. So maximum likelihood is all we have. 2440

2441 When the labels are observed, we can estimate the parameters of the Naïve Bayes 2442 probability model separately for each label. But marginalizing over the labels couples 2443 these parameters, making direct optimization of $\log p(\mathbf{x})$ intractable. We will approximate 2444 the log-likelihood by introducing an *auxiliary variable* $\mathbf{q}^{(i)}$, which is a distribution over the 2445 label set $\mathcal{Z} = \{1, 2, \dots, K\}$. The optimization procedure will alternate between updates to 2446 \mathbf{q} and updates to the parameters $(\boldsymbol{\phi}, \boldsymbol{\mu})$. Thus, $\mathbf{q}^{(i)}$ plays here as in soft K -means.

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

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

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

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

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

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

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

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

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

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

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,

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

2465 The term $\log p(\mathbf{x}^{(i)} | z; \boldsymbol{\phi})$ is the conditional log-likelihood for the multinomial, which
2466 expands to,

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

2467 where C is a constant with respect to $\boldsymbol{\phi}$ — see Equation 2.12 in § 2.1 for more discussion
2468 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,

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

2469 where the counter $j \in \{1, 2, \dots, V\}$ indexes over base features, such as words.

2470 This update sets ϕ_z equal to the relative frequency estimate of the *expected counts* under
2471 the distribution q . As in supervised Naïve Bayes, we can smooth these counts by adding
2472 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
2473 expected frequency of cluster z . These probabilities can also be smoothed. In sum, the
2474 M-step is just like Naïve Bayes, but with expected counts rather than observed counts.

2475 The multinomial likelihood $p(\mathbf{x} | z)$ can be replaced with other probability distribu-
2476 tions: for example, for continuous observations, a Gaussian distribution can be used. In
2477 some cases, there is no closed-form update to the parameters of the likelihood. One ap-
2478 proach is to run gradient-based optimization at each M-step; another is to simply take a
2479 single step along the gradient step and then return to the E-step (Berg-Kirkpatrick et al.,
2480 2010).

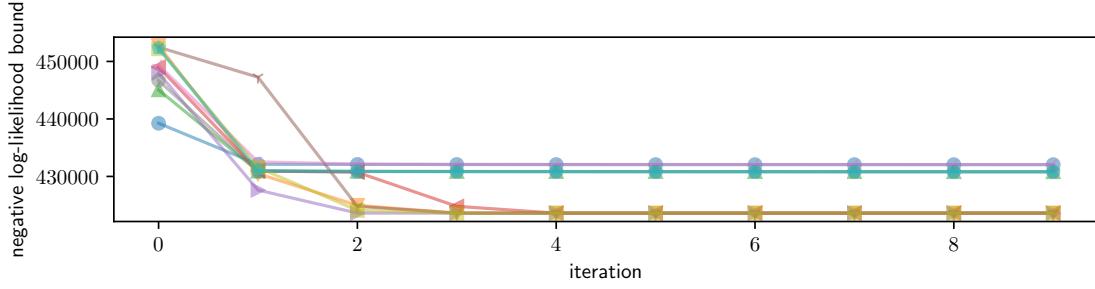


Figure 5.2: Sensitivity of expectation maximization to initialization. Each line shows the progress of optimization from a different random initialization.

2481 5.1.3 EM as an optimization algorithm

2482 Algorithms that alternate between updating subsets of the parameters are called **coordi-**
 2483 **nate ascent** algorithms. The objective J (the lower bound on the marginal likelihood of
 2484 the data) is separately convex in q and (μ, ϕ) , but it is not jointly convex in all terms; this
 2485 condition is known as **biconvexity**. Each step of the expectation-maximization algorithm
 2486 is guaranteed not to decrease the lower bound J , which means that EM will converge
 2487 towards a solution at which no nearby points yield further improvements. This solution
 2488 is a **local optimum** — it is as good or better than any of its immediate neighbors, but is
 2489 *not* guaranteed to be optimal among all possible configurations of (q, μ, ϕ) .

2490 The fact that there is no guarantee of global optimality means that initialization is
 2491 important: where you start can determine where you finish. To illustrate this point,
 2492 Figure 5.2 shows the objective function for EM with ten different random initializations:
 2493 while the objective function improves monotonically in each run, it converges to several
 2494 different values.¹ For the convex objectives that we encountered in chapter 2, it was not
 2495 necessary to worry about initialization, because gradient-based optimization guaranteed
 2496 to reach the global minimum. But in expectation-maximization — and in the deep neural
 2497 networks from chapter 3 — initialization matters.

2498 In **hard EM**, each $q^{(i)}$ distribution assigns probability of 1 to a single label $\hat{z}^{(i)}$, and zero
 2499 probability to all others (Neal and Hinton, 1998). This is similar in spirit to K -means clus-
 2500 tering, and can outperform standard EM in some cases (Spitkovsky et al., 2010). Another
 2501 variant of expectation maximization incorporates stochastic gradient descent (SGD): after
 2502 performing a local E-step at each instance $x^{(i)}$, we immediately make a gradient update
 2503 to the parameters (μ, ϕ) . This algorithm has been called **incremental expectation maxi-**
 2504 **mization** (Neal and Hinton, 1998) and **online expectation maximization** (Sato and Ishii,
 2505 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.

2506 optimum for the likelihood $p(\mathbf{x} \mid z)$, and in online settings where new data is constantly
 2507 streamed in (see Liang and Klein, 2009, for a comparison for online EM variants).

2508 **5.1.4 How many clusters?**

2509 So far, we have assumed that the number of clusters K is given. In some cases, this as-
 2510 sumption is valid. For example, a lexical semantic resource like WordNet might define the
 2511 number of senses for a word. In other cases, the number of clusters could be a parameter
 2512 for the user to tune: some readers want a coarse-grained clustering of news stories into
 2513 three or four clusters, while others want a fine-grained clustering into twenty or more.
 2514 But many times there is little extrinsic guidance for how to choose K .

2515 One solution is to choose the number of clusters to maximize a metric of clustering
 2516 quality. The other parameters μ and ϕ are chosen to maximize the log-likelihood bound
 2517 J , so this might seem a potential candidate for tuning K . However, J will never decrease
 2518 with K : if it is possible to obtain a bound of J_K with K clusters, then it is always possible
 2519 to do at least as well with $K + 1$ clusters, by simply ignoring the additional cluster and
 2520 setting its probability to zero in q and μ . It is therefore necessary to introduce a penalty
 2521 for model complexity, so that fewer clusters are preferred. For example, the Akaike Infor-
 2522 mation Crition (AIC; Akaike, 1974) is the linear combination of the number of parameters
 2523 and the log-likelihood,

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

2524 where M is the number of parameters. In an expectation-maximization clustering algo-
 2525 rithm, $M = K \times V + K$. Since the number of parameters increases with the number of
 2526 clusters K , the AIC may prefer more parsimonious models, even if they do not fit the data
 2527 quite as well.

2528 Another choice is to maximize the **predictive likelihood** on heldout data. This data
 2529 is not used to estimate the model parameters ϕ and μ , and so it is not the case that the
 2530 likelihood on this data is guaranteed to increase with K . Figure 5.3 shows the negative
 2531 log-likelihood on training and heldout data, as well as the AIC.

2532 ***Bayesian nonparametrics** An alternative approach is to treat the number of clusters as
 2533 another latent variable. This requires statistical inference over a set of models with a vari-
 2534 able number of clusters. This is not possible within the framework of expectation max-
 2535 imization, but there are several alternative inference procedures which can be applied,
 2536 including **Markov Chain Monte Carlo (MCMC)**, which is briefly discussed in § 5.5 (for
 2537 more details, see Chapter 25 of Murphy, 2012). Bayesian nonparametrics have been ap-
 2538 plied to the problem of unsupervised word sense induction, learning not only the word
 2539 senses but also the number of senses per word (Reisinger and Mooney, 2010).

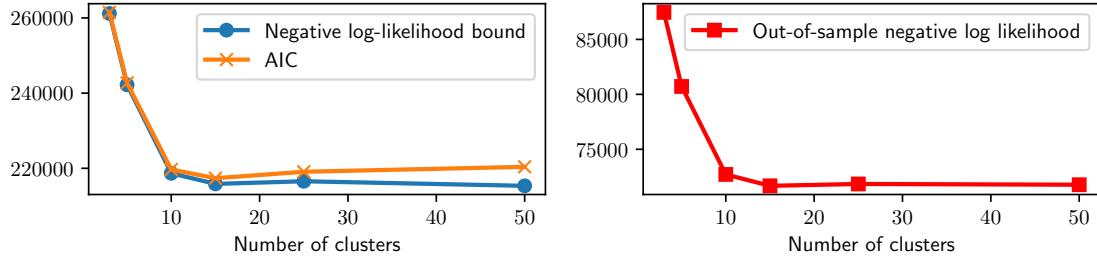


Figure 5.3: The negative log-likelihood and AIC for several runs of expectation maximization, on synthetic data. Although the data was generated from a model with $K = 10$, the optimal number of clusters is $\hat{K} = 15$, according to AIC and the heldout log-likelihood. The training set log-likelihood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

2540 5.2 Applications of expectation-maximization

2541 EM is not really an “algorithm” like, say, quicksort. Rather, it is a framework for learning
2542 with missing data. The recipe for using EM on a problem of interest is:

- 2543 • Introduce latent variables z , such that it is easy to write the probability $P(x, z)$. It
2544 should also be easy to estimate the associated parameters, given knowledge of z .
- 2545 • Derive the E-step updates for $q(z)$, which is typically factored as $q(z) = \prod_{i=1}^N q_{z(i)}(z^{(i)})$,
2546 where i is an index over instances.
- 2547 • The M-step updates typically correspond to the soft version of a probabilistic super-
2548 vised learning algorithm, like Naïve Bayes.

2549 This section discusses a few of the many applications of this general framework.

2550 5.2.1 Word sense induction

2551 The chapter began by considering the problem of word sense disambiguation when the
2552 senses are not known in advance. Expectation-maximization can be applied to this prob-
2553 lem by treating each cluster as a word sense. Each instance represents the use of an
2554 ambiguous word, and $x^{(i)}$ is a vector of counts for the other words that appear nearby:
2555 Schütze (1998) uses all words within a 50-word window. The probability $p(x^{(i)} | z)$ can be
2556 set to the multinomial distribution, as in Naïve Bayes. The EM algorithm can be applied
2557 directly to this data, yielding clusters that (hopefully) correspond to the word senses.

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

(c) Jacob Eisenstein 2018. Work in progress.

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

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

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

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

5.2.2 Semi-supervised learning

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

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

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

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

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

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

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

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

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

2577 The left sum is identical to the objective in Naïve Bayes; the right sum is the marginal log-
 2578 likelihood for expectation-maximization clustering, from Equation 5.5. We can construct a
 2579 lower bound on this log-likelihood by introducing distributions $q^{(j)}$ for all $j \in \{N_\ell + 1, \dots, N_\ell + N_u\}$.
 2580 The E-step updates these distributions; the M-step updates the parameters ϕ and μ , us-
 2581 ing the expected counts from the unlabeled data and the observed counts from the labeled
 2582 data.

2583 A critical issue in semi-supervised learning is how to balance the impact of the labeled
 2584 and unlabeled data on the classifier weights, especially when the unlabeled data is much
 2585 larger than the labeled dataset. The risk is that the unlabeled data will dominate, caus-
 2586 ing the parameters to drift towards a “natural clustering” of the instances — which may
 2587 not correspond to a good classifier for the labeled data. One solution is to heuristically
 2588 reweight the two components of Equation 5.26, tuning the weight of the two components
 2589 on a heldout development set (Nigam et al., 2000).

2590 **5.2.3 Multi-component modeling**

2591 As a final application, let’s return to fully supervised classification. A classic dataset for
 2592 text classification is 20 newsgroups, which contains posts to a set of online forums, called
 2593 newsgroups. One of the newsgroups is `comp.sys.mac.hardware`, which discusses Ap-
 2594 ple computing hardware. Suppose that within this newsgroup there are two kinds of
 2595 posts: reviews of new hardware, and question-answer posts about hardware problems.
 2596 The language in these *components* of the `mac.hardware` class might have little in com-
 2597 mon; if so, it would be better to model these components separately, rather than treating
 2598 their union as a single class. However, the component responsible for each instance is not
 2599 directly observed.

2600 Recall that Naïve Bayes is based on a generative process, which provides a stochastic
 2601 explanation for the observed data. In Naïve Bayes, each label is drawn from a categorical
 2602 distribution with parameter μ , and each vector of word counts is drawn from a multi-
 2603 nomial distribution with parameter ϕ_y . For multi-component modeling, we envision a
 2604 slightly different generative process, incorporating both the observed label $y^{(i)}$ and the
 2605 latent component $z^{(i)}$. This generative process is shown in Algorithm 9. A new parameter
 2606 $\beta_{y^{(i)}}$ defines the distribution of components, conditioned on the label $y^{(i)}$. The component,
 2607 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,

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

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

2608 **5.3 Semi-supervised learning**

2609 In semi-supervised learning, the learner makes use of both labeled and unlabeled data.
 2610 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

2647 **Co-training** is an iterative multi-view learning algorithm, in which there are separate
 2648 classifiers for each view (Blum and Mitchell, 1998). At each iteration of the algorithm, each
 2649 classifier predicts labels for a subset of the unlabeled instances, using only the features
 2650 available in its view. These predictions are then used as ground truth to train the classifiers
 2651 associated with the other views. In the example shown in Table 5.2, the classifier on $\mathbf{x}^{(1)}$
 2652 might correctly label instance #5 as a person, because of the feature *Dr*; this instance would
 2653 then serve as training data for the classifier on $\mathbf{x}^{(2)}$, which would then be able to correctly
 2654 label instance #6, thanks to the feature *recommended*. If the views are truly independent,
 2655 this procedure is robust to drift. Furthermore, it imposes no restrictions on the classifiers
 2656 that can be used for each view.

2657 Word-sense disambiguation is particularly suited to multi-view learning, thanks to the
 2658 heuristic of “one sense per discourse”: if a polysemous word is used more than once in
 2659 a given text or conversation, all usages refer to the same sense (Gale et al., 1992). This
 2660 motivates a multi-view learning approach, in which one view corresponds to the local
 2661 context (the surrounding words), and another view corresponds to the global context at
 2662 the document level (Yarowsky, 1995). The local context view is first trained on a small
 2663 seed dataset. We then identify its most confident predictions on unlabeled instances. The
 2664 global context view is then used to extend these confident predictions to other instances
 2665 within the same documents. These new instances are added to the training data to the
 2666 local context classifier, which is retrained and then applied to the remaining unlabeled
 2667 data.

2668 5.3.2 Graph-based algorithms

2669 Another family of approaches to semi-supervised learning begins by constructing a graph,
 2670 in which pairs of instances are linked with symmetric weights $\omega_{i,j}$, e.g.,

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

2671 The goal is to use this weighted graph to propagate labels from a small set of labeled
 2672 instances to larger set of unlabeled instances.

2673 In **label propagation**, this is done through a series of matrix operations (Zhu et al.,
 2674 2003). Let \mathbf{Q} be a matrix of size $N \times K$, in which each row $\mathbf{q}^{(i)}$ describes the labeling
 2675 of instance i . When ground truth labels are available, then $\mathbf{q}^{(i)}$ is an indicator vector,
 2676 with $q_{y^{(i)}}^{(i)} = 1$ and $q_{y' \neq y^{(i)}}^{(i)} = 0$. Let us refer to the submatrix of rows containing labeled
 2677 instances as \mathbf{Q}_L , and the remaining rows as \mathbf{Q}_U . The rows of \mathbf{Q}_U are initialized to assign
 2678 equal probabilities to all labels, $q_{i,k} = \frac{1}{K}$.

2679 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,

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

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

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

2680 The expression $\tilde{\mathbf{Q}}_U \mathbf{1}$ indicates multiplication of $\tilde{\mathbf{Q}}_U$ by a column vector of ones, which is
 2681 equivalent to computing the sum of each row of $\tilde{\mathbf{Q}}_U$. The matrix $\text{Diag}(\mathbf{s})$ is a diagonal
 2682 matrix with the elements of \mathbf{s} on the diagonals. The product $\text{Diag}(\mathbf{s})^{-1} \tilde{\mathbf{Q}}_U$ has the effect
 2683 of normalizing the rows of $\tilde{\mathbf{Q}}_U$, so that each row of \mathbf{Q}_U is a probability distribution over
 2684 labels.

2685 5.4 Domain adaptation

2686 In many practical scenarios, the labeled data differs in some key respect from the data
 2687 to which the trained model is to be applied. A classic example is in consumer reviews:
 2688 we may have labeled reviews of movies (the **source domain**), but we want to predict the
 2689 reviews of appliances (the **target domain**). A similar issues arise with genre differences:
 2690 most linguistically-annotated data is news text, but application domains range from social
 2691 media to electronic health records. In general, there may be several source and target
 2692 domains, each with their own properties; however, for simplicity, this discussion will
 2693 focus mainly on the case of a single source and target domain.

2694 The simplest approach is “direct transfer”: train a classifier on the source domain,
 2695 and apply it directly to the target domain. The accuracy of this approach depends on the
 2696 extent to which features are shared across domains. In review text, words like *outstanding*

and *disappointing* will apply across both movies and appliances; but others, like *terrifying*, may have meanings that are domain-specific. **Domain adaptation** algorithms attempt to do better than direct transfer, by learning from data in both domains. There are two main families of domain adaptation algorithms, depending on whether any labeled data is available in the target domain.

5.4.1 Supervised domain adaptation

In supervised domain adaptation, there is a small amount of labeled data in the target domain, and a large amount of data in the source domain. The simplest approach would be to ignore domain differences, and simply merge the training data from the source and target domains. There are several other baseline approaches to dealing with this scenario (Daumé III, 2007):

Interpolation. Train a classifier for each domain, and combine their predictions. For example,

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

where Ψ_s and Ψ_t are the scoring functions from the source and target domain classifiers respectively, and λ_s is the interpolation weight.

Prediction. Train a classifier on the source domain data, use its prediction as an additional feature in a classifier trained on the target domain data.

Priors. Train a classifier on the source domain data, and use its weights as a prior distribution on the weights of the classifier for the target domain data. This is equivalent to regularizing the target domain weights towards the weights of the source domain classifier (Chelba and Acero, 2006),

$$\ell(\boldsymbol{\theta}_t) = \sum_{i=1}^N \ell^{(i)}(\mathbf{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}_t) + \lambda \|\boldsymbol{\theta}_t - \boldsymbol{\theta}_s\|_2^2, \quad [5.38]$$

where $\ell^{(i)}$ is the prediction loss on instance i , and λ is the regularization weight.

An effective and “frustratingly simple” alternative is EasyAdapt (Daumé III, 2007), which creates copies of each feature: one for each domain and one for the cross-domain setting. For example, a negative review of the film *Wonder Woman* begins, *As boring and flavorless as a three-day-old grilled cheese sandwich....*⁸ The resulting bag-of-words feature

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

vector would be,

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

with $(boring, -, \text{MOVIE})$ indicating the word *boring* appearing in a negative labeled document in the MOVIE domain, and $(boring, -, *)$ indicating the same word in a negative labeled document in *any* domain. It is up to the learner to allocate weight between the domain-specific and cross-domain features: for words that facilitate prediction in both domains, the learner will use the cross-domain features; for words that are relevant only to a single domain, the domain-specific features will be used. Any discriminative classifier can be used with these augmented features.⁹

5.4.2 Unsupervised domain adaptation

In unsupervised domain adaptation, there is no labeled data in the target domain. Unsupervised domain adaptation algorithms cope with this problem by trying to make the data from the source and target domains as similar as possible. This is typically done by learning a **projection function**, which puts the source and target data in a shared space, in which a learner can generalize across domains. This projection is learned from data in both domains, and is applied to the base features — for example, the bag-of-words in text classification. The projected features can then be used both for training and for prediction.

5.4.2.1 Linear projection

In linear projection, the cross-domain representation is constructed by a matrix-vector product,

$$\mathbf{g}(\mathbf{x}^{(i)}) = \mathbf{U}\mathbf{x}^{(i)}. \quad [5.39]$$

The projected vectors $\mathbf{g}(\mathbf{x}^{(i)})$ can then be used as base features during both training (from the source domain) and prediction (on the target domain).

The projection matrix \mathbf{U} can be learned in a number of different ways, but many approaches focus on compressing and reconstructing the base features (Ando and Zhang, 2005). For example, we can define a set of **pivot features**, which are typically chosen because they appear in both domains: in the case of review documents, pivot features might include evaluative adjectives like *outstanding* and *disappointing* (Blitzer et al., 2007). For each pivot feature j , we define an auxiliary problem of predicting whether the feature is

⁹EasyAdapt can be explained as a hierarchical Bayesian model, in which the weights for each domain are drawn from a shared prior (Finkel and Manning, 2009).

2745 present in each example, using the remaining base features. Let ϕ_j denote the weights of
 2746 this classifier, and us horizontally concatenate the weights for each of the N_p pivot features
 2747 into a matrix $\Phi = [\phi_1, \phi_2, \dots, \phi_{N_p}]$.

2748 We then perform truncated singular value decomposition on Φ , as described in § 5.2.1,
 2749 obtaining $\Phi \approx \mathbf{U}\mathbf{S}\mathbf{V}^\top$. The rows of the matrix \mathbf{U} summarize information about each base
 2750 feature: indeed, the truncated singular value decomposition identifies a low-dimension
 2751 basis for the weight matrix Φ , which in turn links base features to pivot features. Sup-
 2752 pose that a base feature *reliable* occurs only in the target domain of appliance reviews.
 2753 Nonetheless, it will have a positive weight towards some pivot features (e.g., *outstanding*,
 2754 *recommended*), and a negative weight towards others (e.g., *worthless*, *unpleasant*). A base
 2755 feature such as *watchable* might have the same associations with the pivot features, and
 2756 therefore, $\mathbf{u}_{\text{reliable}} \approx \mathbf{u}_{\text{watchable}}$. The matrix \mathbf{U} can thus project the base features into a
 2757 space in which this information is shared.

2758 5.4.2.2 Non-linear projection

2759 Non-linear transformations of the base features can be accomplished by implementing
 2760 the transformation function as a deep neural network, which is trained from an auxiliary
 2761 objective.

2762 **Denoising objectives** One possibility is to train a projection function to reconstruct a
 2763 corrupted version of the original input. The original input can be corrupted in various
 2764 ways: by the addition of random noise (Glorot et al., 2011; Chen et al., 2012), or by the
 2765 deletion of features (Chen et al., 2012; Yang and Eisenstein, 2015). Denoising objectives
 2766 share many properties of the linear projection method described above: they enable the
 2767 projection function to be trained on large amounts of unlabeled data from the target do-
 2768 main, and allow information to be shared across the feature space, thereby reducing sen-
 2769 sitivity to rare and domain-specific features.

2770 **Adversarial objectives** The ultimate goal is for the transformed representations $\mathbf{g}(\mathbf{x}^{(i)})$
 2771 to be domain-general. This can be made an explicit optimization criterion by comput-
 2772 ing the similarity of transformed instances both within and between domains (Tzeng
 2773 et al., 2015), or by formulating an auxiliary classification task, in which the domain it-
 2774 self is treated as a label (Ganin et al., 2016). This setting is **adversarial**, because we want
 2775 to learn a representation that makes this classifier perform poorly. At the same time, we
 2776 want $\mathbf{g}(\mathbf{x}^{(i)})$ to enable accurate predictions of the labels $y^{(i)}$.

2777 To formalize this idea, let $d^{(i)}$ represent the domain of instance i , and let $\ell_d(\mathbf{g}(\mathbf{x}^{(i)}), d^{(i)}; \theta_d)$
 2778 represent the loss of a classifier (typically a deep neural network) trained to predict $d^{(i)}$
 2779 from the transformed representation $\mathbf{g}(\mathbf{x}^{(i)})$, using parameters θ_d . Analogously, let $\ell_y(\mathbf{g}(\mathbf{x}^{(i)}), y^{(i)}; \theta_y)$
 2780 represent the loss of a classifier trained to predict the label $y^{(i)}$ from $\mathbf{g}(\mathbf{x}^{(i)})$, using param-

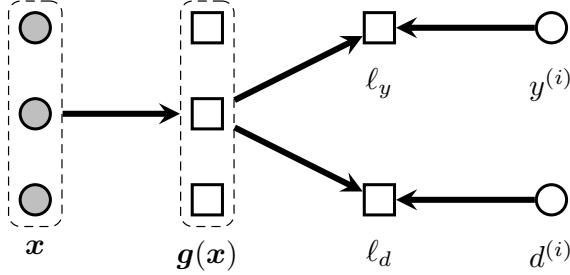


Figure 5.4: A schematic view of adversarial domain adaptation. The loss ℓ_y is computed only for instances from the source domain, where labels $y^{(i)}$ are available.

eters θ_y . The transformation g can then be trained from two criteria: it should yield accurate predictions of the labels $y^{(i)}$, while making *inaccurate* predictions of the domains $d^{(i)}$. This can be formulated as a joint optimization problem,

$$\min_{f, \theta_g, \theta_y, \theta_d} \sum_{i=1}^{N_\ell+N_u} \ell_d(g(\mathbf{x}^{(i)}; \theta_g), d^{(i)}; \theta_d) - \sum_{i=1}^{N_\ell} \ell_y(g(\mathbf{x}^{(i)}), y^{(i)}; \theta_y), \quad [5.40]$$

where N_ℓ is the number of labeled instances and N_u is the number of unlabeled instances, with the labeled instances appearing first in the dataset. This setup is shown in Figure 5.4. The loss can be optimized by stochastic gradient descent, jointly training the parameters of the non-linear transformation θ_g , and the parameters of the prediction models θ_d and θ_y .

5.5 *Other approaches to learning with latent variables

Expectation maximization provides a general approach to learning with latent variables, but it has limitations. One is the sensitivity to initialization; in practical applications, considerable attention may need to be devoted to finding a good initialization. A second issue is that EM tends to be easiest to apply in cases where the latent variables have a clear decomposition (in the cases we have considered, they decompose across the instances). For these reasons, it is worth briefly considering some alternatives to EM.

5.5.1 Sampling

In EM clustering, there is a distribution $q^{(i)}$ for the missing data related to each instance. The M-step consists of updating the parameters of this distribution. An alternative is to draw samples of the latent variables. If the sampling distribution is designed correctly, this procedure will eventually converge to drawing samples from the true posterior over the missing data, $p(z^{(1:N_z)} | \mathbf{x}^{(1:N_x)})$. For example, in the case of clustering, the missing

2802 data $\mathbf{z}^{(1:N_z)}$ is the set of cluster memberships, $\mathbf{y}^{(1:N)}$, so we draw samples from the pos-
 2803 terior distribution over clusterings of the data. If a single clustering is required, we can
 2804 select the one with the highest conditional likelihood, $\hat{\mathbf{z}} = \text{argmax}_{\mathbf{z}} p(\mathbf{z}^{(1:N_z)} | \mathbf{x}^{(1:N_x)})$.

This general family of algorithms is called **Markov Chain Monte Carlo (MCMC)**: “Monte Carlo” because it is based on a series of random draws; “Markov Chain” because the sampling procedure must be designed such that each sample depends only on the previous sample, and not on the entire sampling history. **Gibbs sampling** is an MCMC algorithm in which each latent variable is sampled from its posterior distribution,

$$\mathbf{z}^{(n)} | \mathbf{x}, \mathbf{z}^{(-n)} \sim p(\mathbf{z}^{(n)} | \mathbf{x}, \mathbf{z}^{(-n)}), \quad [5.41]$$

where $\mathbf{z}^{(-n)}$ indicates $\{\mathbf{z} \setminus \mathbf{z}^{(n)}\}$, the set of all latent variables except for $\mathbf{z}^{(n)}$. Repeatedly drawing samples over all latent variables constructs a Markov chain, and which is guaranteed to converge to a sequence of samples from, $p(\mathbf{z}^{(1:N_z)} | \mathbf{x}^{(1:N_x)})$. In probabilistic clustering, the sampling distribution has the following form,

$$p(\mathbf{z}^{(i)} | \mathbf{x}, \mathbf{z}^{(-i)}) = \frac{p(\mathbf{x}^{(i)} | z^{(i)}; \boldsymbol{\phi}) \times p(z^{(i)}; \boldsymbol{\mu})}{\sum_{z=1}^K p(\mathbf{x}^{(i)} | z; \boldsymbol{\phi}) \times p(z; \boldsymbol{\mu})} \quad [5.42]$$

$$\propto \text{Multinomial}(\mathbf{x}^{(i)}; \boldsymbol{\phi}_{z^{(i)}}) \times \boldsymbol{\mu}_{z^{(i)}}. \quad [5.43]$$

2805 In this case, the sampling distribution does not depend on the other instances $\mathbf{x}^{(-i)}, \mathbf{z}^{(-i)}$:
 2806 given the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$, the posterior distribution over each $z^{(i)}$ can be computed
 2807 from $\mathbf{x}^{(i)}$ alone.

2808 In sampling algorithms, there are several choices for how to deal with the parameters.
 2809 One possibility is to sample them too. To do this, we must add them to the generative
 2810 story, by introducing a prior distribution. For the multinomial and categorical parameters
 2811 in the EM clustering model, the **Dirichlet distribution** is a typical choice, since it defines
 2812 a probability on exactly the set of vectors that can be parameters: vectors that sum to one
 2813 and include only non-negative numbers.¹⁰

2814 To incorporate this prior, the generative model must augmented to indicate that each
 2815 $\boldsymbol{\phi}_z \sim \text{Dirichlet}(\boldsymbol{\alpha}_\phi)$, and $\boldsymbol{\mu} \sim \text{Dirichlet}(\boldsymbol{\alpha}_\mu)$. The hyperparameters $\boldsymbol{\alpha}$ are typically set to

¹⁰If $\sum_i^K \theta_i = 1$ and $\theta_i \geq 0$ for all i , then $\boldsymbol{\theta}$ is said to be on the $K - 1$ **simplex**. A Dirichlet distribution with parameter $\boldsymbol{\alpha} \in \mathbb{R}_+^K$ has support over the $K - 1$ simplex,

$$p_{\text{Dirichlet}}(\boldsymbol{\theta} | \boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^K \theta_i^{\alpha_i - 1} \quad [5.44]$$

$$B(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^K \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^K \alpha_i)}, \quad [5.45]$$

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

2816 a constant vector $\alpha = [\alpha, \alpha, \dots, \alpha]$. When α is large, the Dirichlet distribution tends to
 2817 generate vectors that are nearly uniform; when α is small, it tends to generate vectors that
 2818 assign most of their probability mass to a few entries. Given prior distributions over ϕ
 2819 and μ , we can now include them in Gibbs sampling, drawing values for these parameters
 2820 from posterior distributions that are conditioned on the other variables in the model.

2821 Unfortunately, sampling ϕ and μ usually leads to slow convergence, meaning that a
 2822 large number of samples is required before the Markov chain breaks free from the initial
 2823 conditions. The reason is that the sampling distributions for these parameters are tightly
 2824 constrained by the cluster memberships $y^{(i)}$, which in turn are tightly constrained by the
 2825 parameters. There are two solutions that are frequently employed:

- 2826 • **Empirical Bayesian** methods maintain ϕ and μ as parameters rather than latent
 2827 variables. They still employ sampling in the E-step of the EM algorithm, but they
 2828 update the parameters using expected counts that are computed from the samples
 2829 rather than from parametric distributions. This EM-MCMC hybrid is also known
 2830 as Monte Carlo Expectation Maximization (MCEM; Wei and Tanner, 1990), and is
 2831 well-suited for cases in which it is difficult to compute $q^{(i)}$ directly.
- 2832 • In **collapsed Gibbs sampling**, we analytically integrate ϕ and μ out of the model.
 2833 The cluster memberships $y^{(i)}$ are the only remaining latent variable; we sample them
 2834 from the compound distribution,

$$p(y^{(i)} | \mathbf{x}^{(1:N)}, \mathbf{y}^{(-i)}; \alpha_\phi, \alpha_\mu) = \int_{\phi, \mu} p(\phi, \mu | \mathbf{y}^{(-i)}, \mathbf{x}^{(1:N)}; \alpha_\phi, \alpha_\mu) p(y^{(i)} | \mathbf{x}^{(1:N)}, \mathbf{y}^{(-i)}, \phi, \mu) d\phi d\mu. \quad [5.46]$$

2835 For multinomial and Dirichlet distributions, the sampling distribution can be com-
 2836 puted in closed form.

2837 MCMC algorithms are guaranteed to converge to the true posterior distribution over
 2838 the latent variables, but there is no way to know how long this will take. In practice, the
 2839 rate of convergence depends on initialization, just as expectation-maximization depends
 2840 on initialization to avoid local optima. Thus, while Gibbs Sampling and other MCMC
 2841 algorithms provide a powerful and flexible array of techniques for statistical inference in
 2842 latent variable models, they are not a panacea for the problems experienced by EM.

2843 5.5.2 Spectral learning

Another approach to learning with latent variables is based on the **method of moments**, which makes it possible to avoid the problem of non-convex log-likelihood. Write $\bar{\mathbf{x}}^{(i)}$ for the normalized vector of word counts in document i , so that $\bar{\mathbf{x}}^{(i)} = \mathbf{x}^{(i)} / \sum_{j=1}^V x_j^{(i)}$. Then

we can form a matrix of word-word co-occurrence probabilities,

$$\mathbf{C} = \sum_{i=1}^N \bar{\mathbf{x}}^{(i)} (\bar{\mathbf{x}}^{(i)})^\top. \quad [5.47]$$

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

$$E[\mathbf{C}] = \sum_{i=1}^N \sum_{k=1}^K \Pr(Z^{(i)} = k; \boldsymbol{\mu}) \phi_k \phi_k^\top \quad [5.48]$$

$$= \sum_k^K N \mu_k \phi_k \phi_k^\top \quad [5.49]$$

$$= \Phi \text{Diag}(N\mu) \Phi^\top, \quad [5.50]$$

where Φ is formed by horizontally concatenating $\phi_1 \dots \phi_K$, and $\text{Diag}(N\mu)$ indicates a diagonal matrix with values $N\mu_k$ at position (k, k) . Setting \mathbf{C} equal to its expectation gives,

$$\mathbf{C} = \Phi \text{Diag}(N\mu) \Phi^\top, \quad [5.51]$$

2844 which is similar to the eigendecomposition $\mathbf{C} = \mathbf{Q}\Lambda\mathbf{Q}^\top$. This suggests that simply by
 2845 finding the eigenvectors and eigenvalues of \mathbf{C} , we could obtain the parameters ϕ and μ ,
 2846 and this is what motivates the name **spectral learning**.

2847 While moment-matching and eigendecomposition are similar in form, they impose
 2848 different constraints on the solutions: eigendecomposition requires orthonormality, so
 2849 that $\mathbf{Q}\mathbf{Q}^\top = \mathbb{I}$; in estimating the parameters of a text clustering model, we require that μ
 2850 and the columns of Φ are probability vectors. Spectral learning algorithms must therefore
 2851 include a procedure for converting the solution into vectors that are non-negative and
 2852 sum to one. One approach is to replace eigendecomposition (or the related singular value
 2853 decomposition) with non-negative matrix factorization (Xu et al., 2003), which guarantees
 2854 that the solutions are non-negative (Arora et al., 2013).

2855 After obtaining the parameters ϕ and μ , the distribution over clusters can be com-
 2856 puted from Bayes' rule:

$$p(z^{(i)} | \mathbf{x}^{(i)}; \phi, \mu) \propto p(\mathbf{x}^{(i)} | z^{(i)}; \phi) \times p(z^{(i)}; \mu). \quad [5.52]$$

2857 Spectral learning yields provably good solutions without regard to initialization, and can
 2858 be quite fast in practice. However, it is more difficult to apply to a broad family of gen-
 2859 erative models than more generic techniques like EM and Gibbs Sampling. For more on
 2860 applying spectral learning across a range of latent variable models, see Anandkumar et al.
 2861 (2014).

2862 **Additional resources**

2863 There are a number of other learning paradigms that deviate from supervised learning.

- 2864 • **Active learning:** the learner selects unlabeled instances and requests annotations (Set-
- 2865 tles, 2012).
- 2866 • **Multiple instance learning:** labels are applied to bags of instances, with a positive
- 2867 label applied if at least one instance in the bag meets the criterion (Dietterich et al.,
- 2868 1997; Maron and Lozano-Pérez, 1998).
- 2869 • **Constraint-driven learning:** supervision is provided in the form of explicit con-
- 2870 straints on the learner (Chang et al., 2007; Ganchev et al., 2010).
- 2871 • **Distant supervision:** noisy labels are generated from an external resource (Mintz
- 2872 et al., 2009, also see § 17.2.3).
- 2873 • **Multitask learning:** the learner induces a representation that can be used to solve
- 2874 multiple classification tasks (Collobert et al., 2011).
- 2875 • **Transfer learning:** the learner must solve a classification task that differs from the
- 2876 labeled data (Pan and Yang, 2010).

2877 Expectation maximization was introduced by Dempster et al. (1977), and is discussed

2878 in more detail by Murphy (2012). Like most machine learning treatments, Murphy focus

2879 on continuous observations and Gaussian likelihoods, rather than the discrete observa-

2880 tions typically encountered in natural language processing. Murphy (2012) also includes

2881 an excellent chapter on MCMC; for a textbook-length treatment, see Robert and Casella

2882 (2013). For still more on Bayesian latent variable models, see Barber (2012), and for ap-

2883 plications of Bayesian models to natural language processing, see Cohen (2016). Surveys

2884 are available for semi-supervised learning (Zhu and Goldberg, 2009) and domain adapta-

2885 tion (Søgaard, 2013), although both pre-date the current wave of interest in deep learning.

2886 **Exercises**

- 2887 1. Derive the expectation maximization update for the parameter μ in the EM cluster-
- 2888 ing model.
- 2889 2. The expectation maximization lower bound \mathcal{J} is defined in Equation 5.10. Prove
- 2890 that the inverse $-\mathcal{J}$ is convex in q . You can use the following facts about convexity:

 - 2891 • $f(\mathbf{x})$ is convex in \mathbf{x} iff $\alpha f(\mathbf{x}_1) + (1 - \alpha)f(\mathbf{x}_2) \geq f(\alpha\mathbf{x}_1 + (1 - \alpha)\mathbf{x}_2)$ for all
 - 2892 $\alpha \in [0, 1]$.
 - 2893 • If $f(\mathbf{x})$ and $g(\mathbf{x})$ are both convex in \mathbf{x} , then $f(\mathbf{x}) + g(\mathbf{x})$ is also convex in \mathbf{x} .

- 2894 • $\log(x + y) \leq \log x + \log y.$

2895 3. Derive the E-step and M-step updates for the following generative model. You may
 2896 assume that the labels $y^{(i)}$ are observed, but $z_m^{(i)}$ is not.

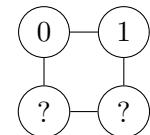
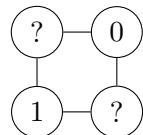
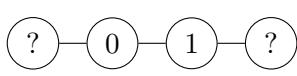
- 2897 • For each instance i ,

- 2898 – Draw label $y^{(i)} \sim \text{Categorical}(\boldsymbol{\mu})$
- 2899 – For each token $m \in \{1, 2, \dots, M^{(i)}\}$
 - 2900 * Draw $z_m^{(i)} \sim \text{Categorical}(\pi)$
 - 2901 * If $z_m^{(i)} = 0$, draw the current token from a label-specific distribution,
 $w_m^{(i)} \sim \phi_{y^{(i)}}$
 - 2903 * If $z_m^{(i)} = 1$, draw the current token from a document-specific distribu-
 $w_m^{(i)} \sim \nu^{(i)}$

2905 4. Use expectation-maximization clustering to train a word-sense induction system,
 2906 applied to the word *say*.

- 2907 • Import `nltk`, run `nltk.download()` and select `semcor`. Import `semcor`
 2908 from `nltk.corpus`.
- 2909 • The command `semcor.tagged_sentences(tag='sense')` returns an iter-
 2910 ator over sense-tagged sentences in the corpus. Each sentence can be viewed as
 2911 an iterator over `tree` objects. For `tree` objects that are sense-annotated words,
 2912 you can access the annotation as `tree.label()`, and the word itself with
 2913 `tree.leaves()`. So `semcor.tagged_sentences(tag='sense')[0][2].label()`
 2914 would return the sense annotation of the third word in the first sentence.
- 2915 • Extract all sentences containing the senses `say.v.01` and `say.v.02`.
- 2916 • Build bag-of-words vectors $x^{(i)}$, containing the counts of other words in those
 2917 sentences, including all words that occur in at least two sentences.
- 2918 • Implement and run expectation-maximization clustering on the merged data.
- 2919 • Compute the frequency with which each cluster includes instances of `say.v.01`
 2920 and `say.v.02`.

2921 5. Using the iterative updates in Equations 5.34-5.36, compute the outcome of the label
 2922 propagation algorithm for the following examples.



(c) Jacob Eisenstein 2018. Work in progress.

2923 The value inside the node indicates the label, $y^{(i)} \in \{0, 1\}$, with $y^{(i)} = ?$ for unlabeled
 2924 nodes. The presence of an edge between two nodes indicates $w_{i,j} = 1$, and the
 2925 absence of an edge indicates $w_{i,j} = 0$. For the third example, you need only compute
 2926 the first three iterations, and then you can guess at the solution in the limit.

2927 In the remaining exercises, you will try out some approaches for semisupervised learning
 2928 and domain adaptation. You will need datasets in multiple domains. You can obtain
 2929 product reviews in multiple domains here: https://www.cs.jhu.edu/~mdredze/datasets/sentiment/processed_acl.tar.gz. Choose a source and target domain,
 2930 e.g. dvds and books, and divide the data for the target domain into training and test sets
 2931 of equal size.

- 2932
- 2933 6. First, quantify the cost of cross-domain transfer.
 - 2934 • Train a logistic regression classifier on the source domain training set, and eval-
 2935 uate it on the target domain test set.
 - 2936 • Train a logistic regression classifier on the target domain training set, and eval-
 2937 uate it on the target domain test set. This is the “direct transfer” baseline.

2938 Compute the difference in accuracy, which is a measure of the transfer loss across
 2939 domains.

- 2940 7. Next, apply the **label propagation** algorithm from § 5.3.2.

2941 As a baseline, using only 5% of the target domain training set, train a classifier, and
 2942 compute its accuracy on the target domain test set.

2943 Next, apply label propagation:

- 2944 • Compute the label matrix \mathbf{Q}_L for the labeled data (5% of the target domain
 2945 training set), with each row equal to an indicator vector for the label (positive
 2946 or negative).
- 2947 • Iterate through the target domain instances, including both test and training
 2948 data. At each instance i , compute all w_{ij} , using Equation 5.32, with $\alpha = 0.01$.
 2949 Use these values to fill in column i of the transition matrix \mathbf{T} , setting all but the
 2950 ten largest values to zero for each column i . Be sure to normalize the column
 2951 so that the remaining values sum to one. You may need to use a sparse matrix
 2952 for this to fit into memory.
- 2953 • Apply the iterative updates from Equations 5.34-5.36 to compute the outcome
 2954 of the label propagation algorithm for the unlabeled examples.

2955 Select the test set instances from \mathbf{Q}_U , and compute the accuracy of this method.
 2956 Compare with the supervised classifier trained only on the 5% sample of the target
 2957 domain training set.

- 2958 8. Using only 5% of the target domain training data (and all of the source domain train-
2959 ing data), implement one of the supervised domain adaptation baselines in § 5.4.1.
2960 See if this improves on the “direct transfer” baseline from the previous problem
- 2961 9. Implement EasyAdapt (§ 5.4.1), again using 5% of the target domain training data
2962 and all of the source domain data.
- 2963 10. Now try unsupervised domain adaptation, using the “linear projection” method
2964 described in § 5.4.2. Specifically:
- 2965 • Identify 500 pivot features as the words with the highest frequency in the (com-
2966 plete) training data for the source and target domains. Specifically, let x_i^d be the
2967 count of the word i in domain d : choose the 500 words with the largest values
2968 of $\min(x_i^{\text{source}}, x_i^{\text{target}})$.
- 2969 • Train a classifier to predict each pivot feature from the remaining words in the
2970 document.
- 2971 • Arrange the features of these classifiers into a matrix Φ , and perform truncated
2972 singular value decomposition, with $k = 20$
- 2973 • Train a classifier from the source domain data, using the combined features
2974 $\mathbf{x}^{(i)} \oplus \mathbf{U}^\top \mathbf{x}^{(i)}$ — these include the original bag-of-words features, plus the pro-
2975 jected features.
- 2976 • Apply this classifier to the target domain test set, and compute the accuracy.

2977

Part II

2978

Sequences and trees

2979

Chapter 6

2980

Language models

2981 In probabilistic classification, the problem is to compute the probability of a label, conditioned
2982 on the text. Let's now consider the inverse problem: computing the probability of
2983 text itself. Specifically, we will consider models that assign probability to a sequence of
2984 word tokens, $p(w_1, w_2, \dots, w_M)$, with $w_m \in \mathcal{V}$. The set \mathcal{V} is a discrete vocabulary,

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

2985 Why would you want to compute the probability of a word sequence? In many applications,
2986 the goal is to produce word sequences as output:

- 2987 • In **machine translation** (chapter 18), we convert from text in a source language to
2988 text in a target language.
- 2989 • In **speech recognition**, we convert from audio signal to text.
- 2990 • In **summarization** (§ 16.3.4.1; § 19.2), we convert from long texts into short texts.
- 2991 • In **dialogue systems** (§ 19.3), we convert from the user's input (and perhaps an
2992 external knowledge base) into a text response.

2993 In many of the systems for performing these tasks, there is a subcomponent that computes
2994 the probability of the output text. The purpose of this component is to generate
2995 texts that are more **fluent**. For example, suppose we want to translate a sentence from
2996 Spanish to English.

2997 (6.1) El cafe negro me gusta mucho.

2998 Here is a literal word-for-word translation (a **gloss**):

2999 (6.2) The coffee black me pleases much.

3000 A good language model of English will tell us that the probability of this translation is
 3001 low, in comparison with more grammatical alternatives,

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

3002 How can we use this fact? Warren Weaver, one of the early leaders in machine trans-
 3003 lation, viewed it as a problem of breaking a secret code (Weaver, 1955):

3004 When I look at an article in Russian, I say: 'This is really written in English,
 3005 but it has been coded in some strange symbols. I will now proceed to decode.'

3006 This observation motivates a generative model (like Naïve Bayes):

3007 • The English sentence $w^{(e)}$ is generated from a **language model**, $p_e(w^{(e)})$.

3008 • The Spanish sentence $w^{(s)}$ is then generated from a **translation model**, $p_{s|e}(w^{(s)} | w^{(e)})$.

Given these two distributions, we can then perform translation by Bayes rule:

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

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

3009 This is sometimes called the **noisy channel model**, because it envisions English text
 3010 turning into Spanish by passing through a noisy channel, $p_{s|e}$. What is the advantage of
 3011 modeling translation this way, as opposed to modeling $p_{e|s}$ directly? The crucial point is
 3012 that the two distributions $p_{s|e}$ (the translation model) and p_e (the language model) can be
 3013 estimated from separate data. The translation model requires examples of correct trans-
 3014 lations, but the language model requires only text in English. Such monolingual data is
 3015 much more widely available. Furthermore, once estimated, the language model p_e can be
 3016 reused in any application that involves generating English text, from summarization to
 3017 speech recognition.

3018 6.1 *N*-gram language models

A simple approach to computing the probability of a sequence of tokens is to use a **relative frequency estimate**. For example, consider the quote, attributed to Picasso, "*computers are useless, they can only give you answers.*" We can estimate the probability of this sentence,

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

3019 This estimator is **unbiased**: in the theoretical limit of infinite data, the estimate will
 3020 be correct. But in practice, we are asking for accurate counts over an infinite number of
 3021 events, since sequences of words can be arbitrarily long. Even with an aggressive upper
 3022 bound of, say, $M = 20$ tokens in the sequence, the number of possible sequences is V^{20} . A
 3023 small vocabulary for English would have $V = 10^4$, so there are 10^{80} possible sequences.
 3024 Clearly, this estimator is very data-hungry, and suffers from high variance: even gram-
 3025 matical sentences will have probability zero if have not occurred in the training data.¹ We
 3026 therefore need to introduce bias to have a chance of making reliable estimates from finite
 3027 training data. The language models that follow in this chapter introduce bias in various
 3028 ways.

We begin with n -gram language models, which compute the probability of a sequence as the product of probabilities of subsequences. The probability of a sequence $p(w) = p(w_1, w_2, \dots, w_M)$ can be refactored using the chain rule (see § A.2):

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

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

Each element in the product is the probability of a word given all its predecessors. We can think of this as a *word prediction* task: given the context *Computers are*, we want to compute a probability over the next token. The relative frequency estimate of the probability of the word *useless* in this context is,

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

3029 We haven't made any approximations yet, and we could have just as well applied the
 3030 chain rule in reverse order,

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

3031 or in any other order. But this means that we also haven't really made any progress:
 3032 to compute the conditional probability $p(w_M | w_{M-1}, w_{M-2}, \dots, w_1)$, we would need to
 3033 model V^{M-1} contexts. Such a distribution cannot be estimated from any realistic sample
 3034 of text.

¹Chomsky has famously argued that this is evidence against the very concept of probabilistic language models: no such model could distinguish the grammatical sentence *colorless green ideas sleep furiously* from the ungrammatical permutation *furiously sleep ideas green colorless*. Indeed, even the bigrams in these two examples are unlikely to occur — at least, not in texts written before Chomsky proposed this example.

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

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

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

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

To compute the probability of an entire sentence, it is convenient to pad the beginning and end with special symbols \square and \blacksquare . Then the bigram ($n = 2$) approximation to the probability of *I like black coffee* is:

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

3035 This model requires estimating and storing the probability of only V^n events, which is
 3036 exponential in the order of the n -gram, and not V^M , which is exponential in the length of
 3037 the sentence. The n -gram probabilities can be computed by relative frequency estimation,

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

3038 The hyperparameter n controls the size of the context used in each conditional proba-
 3039 bility. If this is misspecified, the language model will perform poorly. Let's consider the
 3040 potential problems concretely.

3041 **When n is too small.** Consider the following sentences:

- 3042 (6.3) **Gorillas** always like to groom **their** friends.
 3043 (6.4) The **computer** that's on the 3rd floor of our office building **crashed**.

3044 In each example, the bolded words depend on each other: the likelihood of *their*
 3045 depends on knowing that *gorillas* is plural, and the likelihood of *crashed* depends on
 3046 knowing that the subject is a *computer*. If the n -grams are not big enough to capture
 3047 this context, then the resulting language model would offer probabilities that are too
 3048 low for these sentences, and too high for sentences that fail basic linguistic tests like
 3049 number agreement.

3050 **When n is too big.** In this case, it is hard to get good estimates of the n -gram parameters from
 3051 our dataset, because of data sparsity. To handle the *gorilla* example, it is necessary to
 3052 model 6-grams, which means accounting for V^6 events. Under a very small vocab-
 3053 ularly of $V = 10^4$, this means estimating the probability of 10^{24} distinct events.

3054 These two problems point to another **bias-variance tradeoff** (see § 2.1.4). A small n -
 3055 gram size introduces high bias, and a large n -gram size introduces high variance. But
 3056 in reality we often have both problems at the same time! Language is full of long-range
 3057 dependencies that we cannot capture because n is too small; at the same time, language
 3058 datasets are full of rare phenomena, whose probabilities we fail to estimate accurately
 3059 because n is too large. One solution is to try to keep n large, while still making low-
 3060 variance estimates of the underlying parameters. To do this, we will introduce a different
 3061 sort of bias: **smoothing**.

3062 6.2 Smoothing and discounting

3063 Limited data is a persistent problem in estimating language models. In § 6.1, we presented
 3064 n -grams as a partial solution. sparse data can be a problem even for low-order n -grams;
 3065 at the same time, many linguistic phenomena, like subject-verb agreement, cannot be in-
 3066 corporated into language models without high-order n -grams. It is therefore necessary to
 3067 add additional inductive biases to n -gram language models. This section covers some of
 3068 the most intuitive and common approaches, but there are many more (Chen and Good-
 3069 man, 1999).

3070 6.2.1 Smoothing

3071 A major concern in language modeling is to avoid the situation $p(w) = 0$, which could
 3072 arise as a result of a single unseen n-gram. A similar problem arose in Naïve Bayes, and
 3073 the solution was **smoothing**: adding imaginary “pseudo” counts. The same idea can be
 3074 applied to n -gram language models, as shown here in the bigram case,

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

3075 This basic framework is called **Lidstone smoothing**, but special cases have other names:

- 3076 • **Laplace smoothing** corresponds to the case $\alpha = 1$.
- 3077 • **Jeffreys-Perks law** corresponds to the case $\alpha = 0.5$. Manning and Schütze (1999)
 3078 offer more insight on the justifications for this setting.

3079 To maintain normalization, anything that we add to the numerator (α) must also ap-
 3080 pear in the denominator ($V\alpha$). This idea is reflected in the concept of **effective counts**:

$$c_i^* = (c_i + \alpha) \frac{M}{M + V\alpha}, \quad [6.14]$$

	counts	unsmoothed probability	Lidstone smoothing, $\alpha = 0.1$		Discounting, $d = 0.1$	
			effective counts	smoothed probability	effective counts	smoothed probability
<i>impropriety</i>	8	0.4	7.826	0.391	7.9	0.395
<i>offense</i>	5	0.25	4.928	0.246	4.9	0.245
<i>damage</i>	4	0.2	3.961	0.198	3.9	0.195
<i>deficiencies</i>	2	0.1	2.029	0.101	1.9	0.095
<i>outbreak</i>	1	0.05	1.063	0.053	0.9	0.045
<i>infirmity</i>	0	0	0.097	0.005	0.25	0.013
<i>cephalopods</i>	0	0	0.097	0.005	0.25	0.013

Table 6.1: Example of Lidstone smoothing and absolute discounting in a bigram language model, for the context *(alleged, -)*, for a toy corpus with a total of twenty counts over the seven words shown. Note that discounting decreases the probability for all but the unseen words, while Lidstone smoothing increases the effective counts and probabilities for *deficiencies* and *outbreak*.

where c_i is the count of event i , c_i^* is the effective count, and $M = \sum_{i=1}^V c_i$ is the total number of tokens in the dataset (w_1, w_2, \dots, w_M) . This term ensures that $\sum_{i=1}^V c_i^* = \sum_{i=1}^V c_i = M$. The **discount** for each n-gram is then computed as,

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

3081 6.2.2 Discounting and backoff

3082 Discounting “borrows” probability mass from observed n -grams and redistributes it. In
 3083 Lidstone smoothing, the borrowing is done by increasing the denominator of the relative
 3084 frequency estimates. The borrowed probability mass is then redistributed by increasing
 3085 the numerator for all n -grams. Another approach would be to borrow the same amount
 3086 of probability mass from all observed n -grams, and redistribute it among only the unob-
 3087 served n -grams. This is called **absolute discounting**. For example, suppose we set an
 3088 absolute discount $d = 0.1$ in a bigram model, and then redistribute this probability mass
 3089 equally over the unseen words. The resulting probabilities are shown in Table 6.1.

Discounting reserves some probability mass from the observed data, and we need not redistribute this probability mass equally. Instead, we can **backoff** to a lower-order language model: if you have trigrams, use trigrams; if you don’t have trigrams, use bigrams; if you don’t even have bigrams, use unigrams. This is called **Katz backoff**. In the simple

case of backing off from bigrams to unigrams, the bigram probabilities are computed as,

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

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

3090 The term $\alpha(j)$ indicates the amount of probability mass that has been discounted for
 3091 context j . This probability mass is then divided across all the unseen events, $\{i' : c(i', j) =$
 3092 $0\}$, proportional to the unigram probability of each word i' . The discount parameter d can
 3093 be optimized to maximize performance (typically held-out log-likelihood) on a develop-
 3094 ment set.

3095 6.2.3 *Interpolation

3096 Backoff is one way to combine different order n -gram models. An alternative approach
 3097 is **interpolation**: setting the probability of a word in context to a weighted sum of its
 3098 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,

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

3099 In this equation, p_n^* is the unsmoothed empirical probability given by an n -gram lan-
 3100 guage model, and λ_n is the weight assigned to this model. To ensure that the interpolated
 3101 $p(w)$ is still a valid probability distribution, the values of λ must obey the constraint,
 3102 $\sum_{n=1}^{n_{\max}} \lambda_n = 1$. But how to find the specific values?

3103 An elegant solution is **expectation maximization**. Recall from chapter 5 that we can
 3104 think about EM as learning with *missing data*: we just need to choose missing data such
 3105 that learning would be easy if it weren't missing. What's missing in this case? Think of
 3106 each word w_m as drawn from an n -gram of unknown size, $z_m \in \{1 \dots n_{\max}\}$. This z_m is
 3107 the missing data that we are looking for. Therefore, the application of EM to this problem
 3108 involves the following **generative process**:

3109 **for** Each token $w_m, m = 1, 2, \dots, M$ **do**:
 3110 draw the n -gram size $z_m \sim \text{Categorical}(\lambda)$;
 3112 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 ,

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

3113 A solution is obtained by iterating between updates to q and λ . The complete algorithm
 3114 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  $\lambda$ 
  
```

3115 **6.2.4 *Kneser-Ney smoothing**

3116 Kneser-Ney smoothing is based on absolute discounting, but it redistributes the result-
 3117 ing probability mass in a different way from Katz backoff. Empirical evidence points
 3118 to Kneser-Ney smoothing as the state-of-art for n -gram language modeling (Goodman,
 3119 2001). To motivate Kneser-Ney smoothing, consider the example: *I recently visited ..*
 3120 Which of the following is more likely?

- 3121 • *Francisco*
 3122 • *Duluth*

3123 Now suppose that both bigrams *visited Duluth* and *visited Francisco* are unobserved in
 3124 the training data, and furthermore, the unigram probability $p_1^*(\text{Francisco})$ is greater than
 3125 $p_1^*(\text{Duluth})$. Nonetheless we would still guess that $p(\text{visited Duluth}) > p(\text{visited Francisco})$,
 3126 because *Duluth* is a more “versatile” word: it can occur in many contexts, while *Francisco*
 3127 usually occurs in a single context, following the word *San*. This notion of versatility is the
 3128 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,

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

3129 This is the amount of probability mass left to account for versatility, which we define via
 3130 the *continuation probability* $p_{\text{continuation}}(w)$ as proportional to the number of observed con-
 3131 texts in which w appears. The numerator of the continuation probability is the number of
 3132 contexts u in which w appears; the denominator normalizes the probability by summing
 3133 the same quantity over all words w' .

3134 The idea of modeling versatility by counting contexts may seem heuristic, but there is
 3135 an elegant theoretical justification from Bayesian nonparametrics (Teh, 2006). Kneser-Ney
 3136 smoothing on n -grams was the dominant language modeling technique before the arrival
 3137 of neural language models.

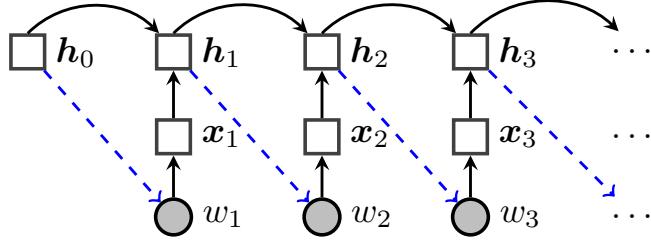


Figure 6.1: The recurrent neural network language model, viewed as an “unrolled” computation graph. Solid lines indicate direct computation, dotted blue lines indicate probabilistic dependencies, circles indicate random variables, and squares indicate computation nodes.

3138 6.3 Recurrent neural network language models

3139 N -gram language models have been largely supplanted by **neural networks**. These mod-
 3140 els do not make the n -gram assumption of restricted context; indeed, they can incorpo-
 3141 ratrarily distant contextual information, while remaining computationally and statisti-
 3142 cally tractable.

3143 The first insight behind neural language models is to treat word prediction as a *dis-
 3144 criminative* learning task.² The goal is to compute the probability $p(w | u)$, where $w \in \mathcal{V}$ is
 3145 a word, and u is the context, which depends on the previous words. Rather than directly
 3146 estimating the word probabilities from (smoothed) relative frequencies, we can treat
 3147 language modeling as a machine learning problem, and estimate parameters that maxi-
 3148 mize the log conditional probability of a corpus.

3149 The second insight is to reparametrize the probability distribution $p(w | u)$ as a func-
 3150 tion of two dense K -dimensional numerical vectors, $\beta_w \in \mathbb{R}^K$, and $v_u \in \mathbb{R}^K$,

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

3151 where $\beta_w \cdot v_u$ represents a dot product. As usual, the denominator ensures that the prob-
 3152 ability distribution is properly normalized. This vector of probabilities is equivalent to
 3153 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).

3180 **6.3.1 Backpropagation through time**

3181 The recurrent neural network language model has the following parameters:

- 3182 • $\phi_i \in \mathbb{R}^K$, the “input” word vectors (these are sometimes called **word embeddings**,
3183 since each word is embedded in a K -dimensional space);
- 3184 • $\beta_i \in \mathbb{R}^K$, the “output” word vectors;
- 3185 • $\Theta \in \mathbb{R}^{K \times K}$, the recurrence operator;
- 3186 • \mathbf{h}_0 , the initial state.

3187 Each of these parameters can be estimated by formulating an objective function over the
3188 training corpus, $L(\mathbf{w})$, and then applying **backpropagation** to obtain gradients on the
3189 parameters from a minibatch of training examples (see § 3.3.1). Gradient-based updates
3190 can be computed from an online learning algorithm such as stochastic gradient descent
3191 (see § 2.5.2).

3192 The application of backpropagation to recurrent neural networks is known as **back-**
3193 **propagation through time**, because the gradients on units at time m depend in turn on the
3194 gradients of units at earlier times $n < m$. Let ℓ_{m+1} represent the negative log-likelihood
3195 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 :

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

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

3196 where g' is the local derivative of the nonlinear function g . The key point in this equation
3197 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
3198 so on, until reaching the initial state \mathbf{h}_0 .

3199 Each derivative $\frac{\partial \mathbf{h}_m}{\partial \theta_{k,k'}}$ will be reused many times: it appears in backpropagation from
3200 the loss ℓ_m , but also in all subsequent losses $\ell_{n>m}$. Neural network toolkits such as
3201 Torch (Collobert et al., 2011) and DyNet (Neubig et al., 2017) compute the necessary

3202 derivatives automatically, and cache them for future use. An important distinction from
 3203 the feedforward neural networks considered in chapter 3 is that the size of the computa-
 3204 tion graph is not fixed, but varies with the length of the input. This poses difficulties for
 3205 toolkits that are designed around static computation graphs, such as TensorFlow (Abadi
 3206 et al., 2016).⁴

3207 **6.3.2 Hyperparameters**

3208 The RNN language model has several hyperparameters that must be tuned to ensure good
 3209 performance. The model capacity is controlled by the size of the word and context vectors
 3210 K , which play a role that is somewhat analogous to the size of the n -gram context. For
 3211 datasets that are large with respect to the vocabulary (i.e., there is a large token-to-type
 3212 ratio), we can afford to estimate a model with a large K , which enables more subtle dis-
 3213 tinctions between words and contexts. When the dataset is relatively small, then K must
 3214 be smaller too, or else the model may “memorize” the training data, and fail to generalize.
 3215 Unfortunately, this general advice has not yet been formalized into any concrete formula
 3216 for choosing K , and trial-and-error is still necessary. Overfitting can also be prevented by
 3217 **dropout**, which involves randomly setting some elements of the computation to zero (Sri-
 3218 vastava et al., 2014), forcing the learner not to rely too much on any particular dimension
 3219 of the word or context vectors. The dropout rate must also be tuned on development data.

3220 **6.3.3 Gated recurrent neural networks**

3221 In principle, recurrent neural networks can propagate information across infinitely long
 3222 sequences. But in practice, repeated applications of the nonlinear recurrence function
 3223 causes this information to be quickly attenuated. The same problem affects learning: back-
 3224 propagation can lead to **vanishing gradients** that decay to zero, or **exploding gradients**
 3225 that increase towards infinity (Bengio et al., 1994). The exploding gradient problem can
 3226 be addressed by clipping gradients at some maximum value (Pascanu et al., 2013). The
 3227 other issues must be addressed by altering the model itself.

3228 The **long short-term memory (LSTM)** (Hochreiter and Schmidhuber, 1997) is a popular
 3229 variant of RNNs that is more robust to these problems. This model augments the hidden
 3230 state \mathbf{h}_m with a **memory cell** c_m . The value of the memory cell at each time m is a gated
 3231 sum of two quantities: its previous value c_{m-1} , and an “update” \tilde{c}_m , which is computed
 3232 from the current input x_m and the previous hidden state \mathbf{h}_{m-1} . The next state \mathbf{h}_m is then
 3233 computed from the memory cell. Because the memory cell is not passed through a non-
 3234 linear squashing function during the update, it is possible for information to propagate
 3235 through the network over long distances.

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

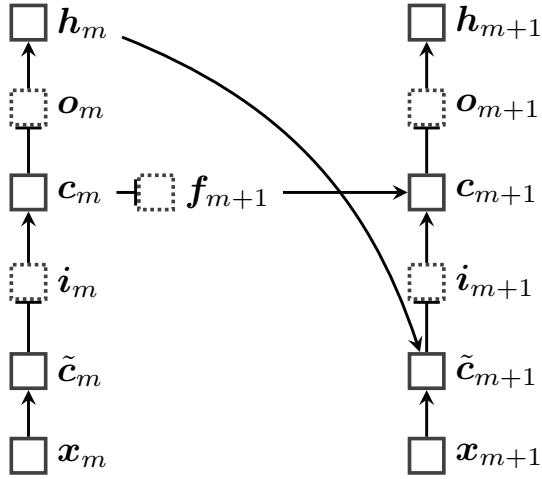


Figure 6.2: The long short-term memory (LSTM) architecture. Gates are shown in boxes with dotted edges. In an LSTM language model, each h_m would be used to predict the next word w_{m+1} .

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:

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

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

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

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

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

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

3236 The operator \odot is an elementwise (Hadamard) product. Each gate is controlled by a vec-
 3237 tor of weights, which parametrize the previous hidden state (e.g., $\Theta^{(h \rightarrow f)}$) and the current
 3238 input (e.g., $\Theta^{(x \rightarrow f)}$), plus a vector offset (e.g., b_f). The overall operation can be infor-
 3239 mally summarized as $(h_m, c_m) = \text{LSTM}(x_m, (h_{m-1}, c_{m-1}))$, with (h_m, c_m) representing
 3240 the LSTM state after reading token m .

3241 The LSTM outperforms standard recurrent neural networks across a wide range of
 3242 problems. It was first used for language modeling by Sundermeyer et al. (2012), but can
 3243 be applied more generally: the vector h_m can be treated as a complete representation of

3244 the input sequence up to position m , and can be used for any labeling task on a sequence
 3245 of tokens, as we will see in the next chapter.

3246 There are several LSTM variants, of which the Gated Recurrent Unit (Cho et al., 2014)
 3247 is one of the more well known. Many software packages implement a variety of RNN
 3248 architectures, so choosing between them is simple from a user’s perspective. Jozefowicz
 3249 et al. (2015) provide an empirical comparison of various modeling choices circa 2015.

3250 6.4 Evaluating language models

3251 Language modeling is not usually an application in itself: language models are typically
 3252 components of larger systems, and they would ideally be evaluated **extrinsically**. This
 3253 means evaluating whether the language model improves performance on the application
 3254 task, such as machine translation or speech recognition. But this is often hard to do, and
 3255 depends on details of the overall system which may be irrelevant to language modeling.
 3256 In contrast, **intrinsic evaluation** is task-neutral. Better performance on intrinsic metrics
 3257 may be expected to improve extrinsic metrics across a variety of tasks, but there is always
 3258 the risk of over-optimizing the intrinsic metric. This section discusses some intrinsic met-
 3259 rics, but keep in mind the importance of performing extrinsic evaluations to ensure that
 3260 intrinsic performance gains carry over to the applications that we care about.

3261 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,

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

3262 treating the entire held-out corpus as a single stream of tokens.

3263 Typically, unknown words are mapped to the $\langle \text{UNK} \rangle$ token. This means that we have
 3264 to estimate some probability for $\langle \text{UNK} \rangle$ on the training data. One way to do this is to fix
 3265 the vocabulary \mathcal{V} to the $V - 1$ words with the highest counts in the training data, and then
 3266 convert all other tokens to $\langle \text{UNK} \rangle$. Other strategies for dealing with out-of-vocabulary
 3267 terms are discussed in § 6.5.

3268 **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,

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

3269 where M is the total number of tokens in the held-out corpus.

3270 Lower perplexities correspond to higher likelihoods, so lower scores are better on this
3271 metric — it is better to be less perplexed. Here are some special cases:

- 3272 • In the limit of a perfect language model, probability 1 is assigned to the held-out
3273 corpus, with $\text{Perplex}(\mathbf{w}) = 2^{-\frac{1}{M} \log_2 1} = 2^0 = 1$.
- 3274 • In the opposite limit, probability zero is assigned to the held-out corpus, which cor-
3275 responds to an infinite perplexity, $\text{Perplex}(\mathbf{w}) = 2^{-\frac{1}{M} \log_2 0} = 2^\infty = \infty$.
- 3276 • Assume a uniform, unigram model in which $p(w_i) = \frac{1}{V}$ for all words in the vocab-
3277 uary. Then,

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

3276 This is the “worst reasonable case” scenario, since you could build such a language
3277 model without even looking at the data.

3278 In practice, language models tend to give perplexities in the range between 1 and V .
3279 A small benchmark dataset is the **Penn Treebank**, which contains roughly a million to-
3280 kens; its vocabulary is limited to 10,000 words, with all other tokens mapped a special
3281 $\langle \text{UNK} \rangle$ symbol. On this dataset, a well-smoothed 5-gram model achieves a perplexity of
3282 141 (Mikolov and Zweig, Mikolov and Zweig), and an LSTM language model achieves
3283 perplexity of roughly 80 (Zaremba, Sutskever, and Vinyals, Zaremba et al.). Various en-
3284 hancements to the LSTM architecture can bring the perplexity below 60 (Merity et al.,
3285 2018). A larger-scale language modeling dataset is the 1B Word Benchmark (Chelba et al.,
3286 2013), which contains text from Wikipedia. On this dataset, a perplexities of around 25
3287 can be obtained by averaging together multiple LSTM language models (Jozefowicz et al.,
3288 2016).

3289 **6.5 Out-of-vocabulary words**

3290 So far, we have assumed a **closed-vocabulary** setting — the vocabulary \mathcal{V} is assumed to be
 3291 a finite set. In realistic application scenarios, this assumption may not hold. Consider, for
 3292 example, the problem of translating newspaper articles. The following sentence appeared
 3293 in a Reuters article on January 6, 2017:⁵

3294 The report said U.S. intelligence agencies believe Russian military intelligence,
 3295 the **GRU**, used intermediaries such as **WikiLeaks**, **DCLeaks.com** and the **Guc-**
 3296 **cifer** 2.0 "persona" to release emails...

3297 Suppose that you trained a language model on the Gigaword corpus,⁶ which was released
 3298 in 2003. The bolded terms either did not exist at this date, or were not widely known; they
 3299 are unlikely to be in the vocabulary. The same problem can occur for a variety of other
 3300 terms: new technologies, previously unknown individuals, new words (e.g., *hashtag*), and
 3301 numbers.

3302 One solution is to simply mark all such terms with a special token, $\langle \text{UNK} \rangle$. While
 3303 training the language model, we decide in advance on the vocabulary (often the K most
 3304 common terms), and mark all other terms in the training data as $\langle \text{UNK} \rangle$. If we do not want
 3305 to determine the vocabulary size in advance, an alternative approach is to simply mark
 3306 the first occurrence of each word type as $\langle \text{UNK} \rangle$.

3307 But it often better to make distinctions about the likelihood of various unknown words.
 3308 This is particularly important in languages that have rich morphological systems, with
 3309 many inflections for each word. For example, Portuguese is only moderately complex
 3310 from a morphological perspective, yet each verb has dozens of inflected forms (see Fig-
 3311 ure 4.3b). In such languages, there will be many word types that we do not encounter in a
 3312 corpus, which are nonetheless predictable from the morphological rules of the language.
 3313 To use a somewhat contrived English example, if *transfenestrate* is in the vocabulary, our
 3314 language model should assign a non-zero probability to the past tense *transfenestrated*,
 3315 even if it does not appear in the training data.

3316 One way to accomplish this is to supplement word-level language models with **character-**
 3317 **level language models**. Such models can use n -grams or RNNs, but with a fixed vocab-
 3318 uary equal to the set of ASCII or Unicode characters. For example Ling et al. (2015)
 3319 propose an LSTM model over characters, and Kim (2014) employ a **convolutional neural**
 3320 **network** (LeCun and Bengio, 1995). A more linguistically motivated approach is to seg-
 3321 ment words into meaningful subword units, known as **morphemes** (see chapter 9). For

⁵Bayoumy, Y. and Strobel, W. (2017, January 6). U.S. intel report: Putin directed cyber 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>

3322 example, Botha and Blunsom (2014) induce vector representations for morphemes, which
3323 they build into a log-bilinear language model; Bhatia et al. (2016) incorporate morpheme
3324 vectors into an LSTM.

3325 Additional resources

3326 A variety of neural network architectures have been applied to language modeling. No-
3327 table earlier non-recurrent architectures include the neural probabilistic language model (Ben-
3328 gio et al., 2003) and the log-bilinear language model (Mnih and Hinton, 2007). Much more
3329 detail on these models can be found in the text by Goodfellow et al. (2016).

3330 Exercises

3331 1. exercises tk

3332 Chapter 7

3333 Sequence labeling

3334 The goal of sequence labeling is to assign tags to words, or more generally, to assign discrete labels to discrete elements in a sequence. There are many applications of sequence labeling in natural language processing, and chapter 8 presents an overview. A classic application is **part-of-speech tagging**, which involves tagging each word by its grammatical category. Coarse-grained grammatical categories include **NOUNs**, which describe things, properties, or ideas, and **VERBs**, which describe actions and events. Consider a simple input:

3341 (7.1) They can fish.

3342 A dictionary of coarse-grained part-of-speech tags might include **NOUN** as the only valid tag for *they*, but both **NOUN** and **VERB** as potential tags for *can* and *fish*. An accurate sequence labeling algorithm should select the verb tag for both *can* and *fish* in (7.1), but it should select the noun tags for the same two words in the phrase *can of fish*.

3346 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:

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

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

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

3347 Here the feature function takes three arguments as input: the sentence to be tagged (e.g., *they can fish*), the proposed tag (e.g., N or V), and the index of the token to which this tag

3349 is applied. This simple feature function then returns a single feature: a tuple including
 3350 the word to be tagged and the tag that has been proposed. If the vocabulary size is V
 3351 and the number of tags is K , then there are $V \times K$ features. Each of these features must
 3352 be assigned a weight. These weights can be learned from a labeled dataset using a clas-
 3353 sification algorithm such as perceptron, but this isn't necessary in this case: it would be
 3354 equivalent to define the classification weights directly, with $\theta_{w,y} = 1$ for the tag y most
 3355 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:

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

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

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

3356 These features contain enough information that a tagger should be able to choose the
 3357 right tag for the word *fish*: words that come after *can* are likely to be verbs, so the feature
 3358 $(w_{m-1} = \text{can}, y_m = \mathbf{V})$ should have a large positive weight.

3359 However, even with this enhanced feature set, it may be difficult to tag some se-
 3360 quences correctly. One reason is that there are often relationships between the tags them-
 3361 selves. For example, in English it is relatively rare for a verb to follow another verb —
 3362 particularly if we differentiate MODAL verbs like *can* and *should* from more typical verbs,
 3363 like *give*, *transcend*, and *befuddle*. We would like to incorporate preferences against tag se-
 3364 quences like VERB-VERB, and in favor of tag sequences like NOUN-VERB. The need for
 3365 such preferences is best illustrated by a **garden path sentence**:

3366 (7.2) The old man the boat.

3367 Grammatically, the word *the* is a DETERMINER. When you read the sentence, what
 3368 part of speech did you first assign to *old*? Typically, this word is an ADJECTIVE — abbrevi-
 3369 ated as J — which is a class of words that modify nouns. Similarly, *man* is usually a noun.
 3370 The resulting sequence of tags is D J N D N. But this is a mistaken “garden path” inter-
 3371 pretation, which ends up leading nowhere. It is unlikely that a determiner would directly

follow a noun,¹ and it is particularly unlikely that the entire sentence would lack a verb. The only possible verb in (7.2) is the word *man*, which can refer to the act of maintaining and piloting something — often boats. But if *man* is tagged as a verb, then *old* is seated between a determiner and a verb, and must be a noun. And indeed, adjectives often have a second interpretation as nouns when used in this way (e.g., *the young*, *the restless*). This reasoning, in which the labeling decisions are intertwined, cannot be applied in a setting where each tag is produced by an independent classification decision.

7.2 Sequence labeling as structure prediction

As an alternative, think of the entire sequence of tags as a label itself. For a given sequence of words $\mathbf{w} = (w_1, w_2, \dots, w_M)$, there is a set of possible taggings $\mathcal{Y}(\mathbf{w}) = \mathcal{Y}^M$, where $\mathcal{Y} = \{\text{N, V, D, ...}\}$ refers to the set of individual tags, and \mathcal{Y}^M refers to the set of tag sequences of length M . We can then treat the sequence labeling problem as a classification problem in the label space $\mathcal{Y}(\mathbf{w})$,

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}(\mathbf{w})}{\operatorname{argmax}} \Psi(\mathbf{w}, \mathbf{y}), \quad [7.7]$$

where $\mathbf{y} = (y_1, y_2, \dots, y_M)$ is a sequence of M tags, and Ψ is a scoring function on pairs of sequences, $V^M \times \mathcal{Y}^M \mapsto \mathbb{R}$. Such a function can include features that capture the relationships between tagging decisions, such as the preference that determiners not follow nouns, or that all sentences have verbs.

Given that the label space is exponentially large in the length of the sequence M , can it ever be practical to perform tagging in this way? The problem of making a series of interconnected labeling decisions is known as **inference**. Because natural language is full of interrelated grammatical structures, inference is a crucial aspect of natural language processing. In English, it is not unusual to have sentences of length $M = 20$; part-of-speech tag sets vary in size from 10 to several hundred. Taking the low end of this range, we have $|\mathcal{Y}(\mathbf{w}_{1:M})| \approx 10^{20}$, one hundred billion billion possible tag sequences. Enumerating and scoring each of these sequences would require an amount of work that is exponential in the sequence length, so inference is intractable.

However, the situation changes when we restrict the scoring function. Suppose we choose a function that decomposes into a sum of local parts,

$$\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m), \quad [7.8]$$

where each $\psi(\cdot)$ scores a local part of the tag sequence. Note that the sum goes up to $M+1$, so that we can include a score for a special end-of-sequence tag, $\psi(\mathbf{w}_{1:M}, \blacklozenge, y_M, M+1)$. We also define a special tag to begin the sequence, $y_0 \triangleq \lozenge$.

¹The main exception occurs with ditransitive verbs, such as *They gave the winner a trophy*.

3401 In a linear model, local scoring function can be defined as a dot product of weights
 3402 and features,

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

3403 The feature vector \mathbf{f} can consider the entire input \mathbf{w} , and can look at pairs of adjacent
 3404 tags. This is a step up from per-token classification: the weights can assign low scores
 3405 to infelicitous tag pairs, such as noun-determiner, and high scores for frequent tag pairs,
 3406 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**):

$$\mathbf{f}(\mathbf{w} = \text{they can fish}, \mathbf{y} = \text{N V V}) = \sum_{m=1}^{M+1} \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.10]$$

$$\begin{aligned} &= \mathbf{f}(\mathbf{w}, \text{N}, \diamond, 1) \\ &\quad + \mathbf{f}(\mathbf{w}, \text{V}, \text{N}, 2) \\ &\quad + \mathbf{f}(\mathbf{w}, \text{V}, \text{V}, 3) \\ &\quad + \mathbf{f}(\mathbf{w}, \blacklozenge, \text{V}, 4) \end{aligned} \quad [7.11]$$

$$\begin{aligned} &= (w_m = \text{they}, y_m = \text{N}) + (y_m = \text{N}, y_{m-1} = \diamond) \\ &\quad + (w_m = \text{can}, y_m = \text{V}) + (y_m = \text{V}, y_{m-1} = \text{N}) \\ &\quad + (w_m = \text{fish}, y_m = \text{V}) + (y_m = \text{V}, y_{m-1} = \text{V}) \\ &\quad + (y_m = \blacklozenge, y_{m-1} = \text{V}). \end{aligned} \quad [7.12]$$

3407 There are seven active features for this example: one for each word-tag pair, and one
 3408 for each tag-tag pair, including a final tag $y_{M+1} = \blacklozenge$. These features capture the two main
 3409 sources of information for part-of-speech tagging in English: which tags are appropriate
 3410 for each word, and which tags tend to follow each other in sequence. Given appropriate
 3411 weights for these features, taggers can achieve high accuracy, even for difficult cases like
 3412 *the old man the boat*. We will now discuss how this restricted scoring function enables
 3413 efficient inference, through the **Viterbi algorithm** (Viterbi, 1967).

³⁴¹⁴ **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:

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} \Psi(\mathbf{w}, \mathbf{y}) \quad [7.13]$$

$$= \operatorname{argmax}_{\mathbf{y}_{1:M}} \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.14]$$

$$= \operatorname{argmax}_{\mathbf{y}_{1:M}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}), \quad [7.15]$$

³⁴¹⁵ where the final line simplifies the notation with the shorthand,

$$s_m(y_m, y_{m-1}) \triangleq \psi(\mathbf{w}_{1:M}, y_m, y_{m-1}, m). \quad [7.16]$$

This inference problem can be solved efficiently using **dynamic programming**, a algorithmic technique for reusing work in recurrent computations. As is often the case in dynamic programming, we begin by solving an auxiliary problem: rather than finding the best tag sequence, we simply compute the *score* of the best tag sequence,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}, \mathbf{y}_{1:M}) = \max_{\mathbf{y}_{1:M}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}). \quad [7.17]$$

This score involves a maximization over all tag sequences of length M , written $\max_{\mathbf{y}_{1:M}}$. This maximization can be broken into two pieces,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}, \mathbf{y}_{1:M}) = \max_{y_M} \max_{\mathbf{y}_{1:M-1}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}), \quad [7.18]$$

which simply says that we maximize over the final tag y_M , and we maximize over all “prefixes”, $\mathbf{y}_{1:M-1}$. But within the sum of scores, only the final term $s_{M+1}(\blacklozenge, y_M)$ depends on y_M . We can pull this term out of the second maximization,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}, \mathbf{y}_{1:M}) = \max_{y_M} s_{M+1}(\blacklozenge, y_M) + \max_{\mathbf{y}_{1:M-1}} \sum_{m=1}^M s_m(y_m, y_{m-1}). \quad [7.19]$$

This same reasoning can be applied recursively to the second term of Equation 7.19, pulling out $s_M(y_M, y_{M-1})$, and so on. We can formalize this idea by defining an auxiliary

Algorithm 11 The Viterbi algorithm. Each $s_m(k, k')$ is a local score for tag $y_m = k$ and $y_{m-1} = k'$.

```

for  $k \in \{0, \dots, K\}$  do
     $v_1(k) = s_1(k, \diamond)$ 
for  $m \in \{2, \dots, M\}$  do
    for  $k \in \{0, \dots, K\}$  do
         $v_m(k) = \max_{k'} s_m(k, k') + v_{m-1}(k')$ 
         $b_m(k) = \operatorname{argmax}_{k'} s_m(k, k') + v_{m-1}(k')$ 
     $y_M = \operatorname{argmax}_k s_{M+1}(\blacklozenge, k) + v_M(k)$ 
    for  $m \in \{M-1, \dots, 1\}$  do
         $y_m = b_m(y_{m+1})$ 
return  $\mathbf{y}_{1:M}$ 
```

Viterbi variable,

$$v_m(y_m) \triangleq \max_{\mathbf{y}_{1:m-1}} \sum_{n=1}^m s_n(y_n, y_{n-1}) \quad [7.20]$$

$$= \max_{y_{m-1}} s_m(y_m, y_{m-1}) + \max_{\mathbf{y}_{1:m-2}} \sum_{n=1}^{m-1} s_n(y_n, y_{n-1}) \quad [7.21]$$

$$= \max_{y_{m-1}} s_m(y_m, y_{m-1}) + v_{m-1}(y_{m-1}). \quad [7.22]$$

3416 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,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}_{1:M}, \mathbf{y}_{1:M}) = v_{M+1}(\blacklozenge). \quad [7.24]$$

3417 Thus, the score of the best labeling for the sequence can be computed in a single forward
 3418 sweep: first compute all variables $v_1(\cdot)$ from Equation 7.23, and then compute all variables
 3419 $v_2(\cdot)$ from the recurrence Equation 7.22, and continue until reaching the final variable
 3420 $v_{M+1}(\blacklozenge)$.

3421 Graphically, it is customary to arrange these variables in a structure known as a **trellis**,
 3422 shown in Figure 7.1. Each column indexes a token m in the sequence, and each row

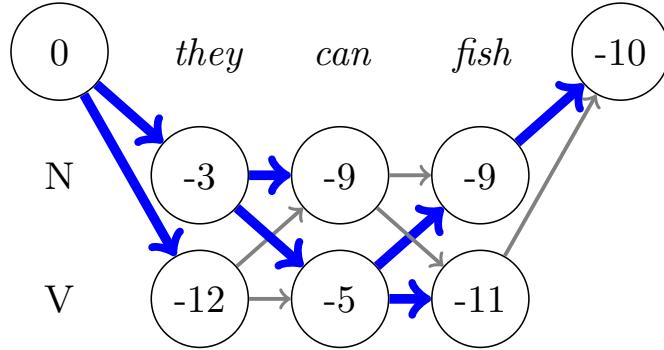


Figure 7.1: The trellis representation of the Viterbi variables, for the example *they can fish*, using the weights shown in Table 7.1.

3423 indexes a tag in \mathcal{Y} ; every $v_{m-1}(k)$ is connected to every $v_m(k')$, that $v_m(k')$ is computed
 3424 from $v_{m-1}(k)$. Special nodes are set aside for the start and end states.

3425 Our real goal is to find the best scoring sequence, not simply to compute its score.
 3426 But solving the auxiliary problem gets us almost all the way there. Recall that each $v_m(k)$
 3427 represents the score of the best tag sequence ending in that tag k in position m . To compute
 3428 this, we maximize over possible values of y_{m-1} . If we keep track of the “argmax” tag that
 3429 maximizes this choice at each step, then we can walk backwards from the final tag, and
 3430 recover the optimal tag sequence. This is indicated in Figure 7.1 by the solid blue lines,
 3431 which we trace back from the final position. These “back-pointers” are written $b_m(k)$,
 3432 indicating the optimal tag y_{m-1} on the path to $Y_m = k$.

3433 The complete Viterbi algorithm is shown in Algorithm 11. When computing the initial
 3434 Viterbi variables $v_1(\cdot)$, we use a special tag, \diamond , to indicate the start of the sequence. When
 3435 computing the final tag Y_M , we use another special tag, \blacklozenge , to indicate the end of the
 3436 sequence. Linguistically, these special tags enable the use of transition features for the tags
 3437 that begin and end the sequence: for example, conjunctions are unlikely to end sentences
 3438 in English, so we would like a low score for $s_{M+1}(\blacklozenge, CC)$; nouns are relatively likely to
 3439 appear at the beginning of sentences, so we would like a high score for $s_1(N, \diamond)$, assuming
 3440 the noun tag is compatible with the first word token w_1 .

3441 **Complexity** If there are K tags and M positions in the sequence, then there are $M \times K$
 3442 Viterbi variables to compute. Computing each variable requires finding a maximum over
 3443 K possible predecessor tags. The total time complexity of populating the trellis is therefore
 3444 $\mathcal{O}(MK^2)$, with an additional factor for the number of active features at each position.
 3445 After completing the trellis, we simply trace the backwards pointers to the beginning of
 3446 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 $-\infty$.3447

7.3.1 Example

3448 Consider the minimal tagset $\{N, V\}$, corresponding to nouns and verbs. Even in this
 3449 tagset, there is considerable ambiguity: for example, the words *can* and *fish* can each take
 3450 both tags. Of the $2 \times 2 \times 2 = 8$ possible taggings for the sentence *they can fish*, four are
 3451 possible given these possible tags, and two are grammatical.²

3452 The values in the trellis in Figure 7.1 are computed from the feature weights defined in
 3453 Table 7.1. We begin with $v_1(N)$, which has only one possible predecessor, the start tag \diamond .
 3454 This score is therefore equal to $s_1(N, \diamond) = -2 - 1 = -3$, which is the sum of the scores for
 3455 the emission and transition features respectively; the backpointer is $b_1(N) = \diamond$. The score
 3456 for $v_1(V)$ is computed in the same way: $s_1(V, \diamond) = -10 - 2 = -12$, and again $b_1(V) = \diamond$.
 3457 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,

$$v_4(\diamond) = \max(v_3(N) + s_4(\diamond, N), v_3(V) + s_4(\diamond, V)) \quad [7.28]$$

$$= \max(-9 + 0 - 1, -11 + 0 - 1) \quad [7.29]$$

$$= -10, \quad [7.30]$$

3458 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.

3459 To compute the optimal tag sequence, we walk backwards from here, next checking
 3460 $b_3(N) = V$, and then $b_2(V) = N$, and finally $b_1(N) = \diamond$. This yields $\mathbf{y} = (N, V, N)$, which
 3461 corresponds to the linguistic interpretation of the fishes being put into cans.

3462 7.3.2 Higher-order features

3463 The Viterbi algorithm was made possible by a restriction of the scoring function to local
 3464 parts that consider only pairs of adjacent tags. We can think of this as a bigram language
 3465 model over tags. A natural question is how to generalize Viterbi to tag trigrams, which
 3466 would involve the following decomposition:

$$\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+2} \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, y_{m-2}, m), \quad [7.31]$$

3467 where $y_{-1} = \diamond$ and $y_{M+2} = \blacklozenge$.

3468 One solution is to create a new tagset $\mathcal{Y}^{(2)}$ from the Cartesian product of the original
 3469 tagset with itself, $\mathcal{Y}^{(2)} = \mathcal{Y} \times \mathcal{Y}$. The tags in this product space are ordered pairs, rep-
 3470 resenting adjacent tags at the token level: for example, the tag (N, V) would represent a
 3471 noun followed by a verb. Transitions between such tags must be consistent: we can have a
 3472 transition from (N, V) to (V, N) (corresponding to the tag sequence $N V N$), but not from
 3473 (N, V) to (N, N) , which would not correspond to any coherent tag sequence. This con-
 3474 straint can be enforced in feature weights, with $\theta_{((a,b),(c,d))} = -\infty$ if $b \neq c$. The remaining
 3475 feature weights can encode preferences for and against various tag trigrams.

3476 In the Cartesian product tag space, there are K^2 tags, suggesting that the time com-
 3477 plexity will increase to $\mathcal{O}(MK^4)$. However, it is unnecessary to max over predecessor tag
 3478 bigrams that are incompatible with the current tag bigram. By exploiting this constraint,
 3479 it is possible to limit the time complexity to $\mathcal{O}(MK^3)$. The space complexity grows to
 3480 $\mathcal{O}(MK^2)$, since the trellis must store all possible predecessors of each tag. In general, the
 3481 time and space complexity of higher-order Viterbi grows exponentially with the order of
 3482 the tag n -grams that are considered in the feature decomposition.

3483 7.4 Hidden Markov Models

3484 Let us now consider how to learn the scores $s_m(y, y')$ that parametrize the Viterbi sequence
 3485 labeling algorithm, beginning with a probabilistic approach. Recall from § 2.1 that the
 3486 probabilistic Naïve Bayes classifier selects the label y to maximize $p(y | \mathbf{x}) \propto p(y, \mathbf{x})$. In
 3487 probabilistic sequence labeling, our goal is similar: select the tag sequence that maximizes
 3488 $p(\mathbf{y} | \mathbf{w}) \propto p(\mathbf{y}, \mathbf{w})$. The locality restriction in Equation 7.8 can be viewed as a conditional
 3489 independence assumption on the random variables \mathbf{y} .

Algorithm 12 Generative process for the hidden Markov model

```

 $y_0 \leftarrow \diamond,$     $m \leftarrow 1$ 
repeat
     $y_m \sim \text{Categorical}(\boldsymbol{\lambda}_{y_{m-1}})$             $\triangleright$  sample the current tag
     $w_m \sim \text{Categorical}(\boldsymbol{\phi}_{y_m})$             $\triangleright$  sample the current word
until  $y_m = \blacklozenge$             $\triangleright$  terminate when the stop symbol is generated

```

3490 Naïve Bayes was introduced as a generative model — a probabilistic story that ex-
 3491 plains the observed data as well as the hidden label. A similar story can be constructed
 3492 for probabilistic sequence labeling: first, the tags are drawn from a prior distribution; next,
 3493 the tokens are drawn from a conditional likelihood. However, for inference to be tractable,
 3494 additional independence assumptions are required. First, the probability of each token
 3495 depends only on its tag, and not on any other element in the sequence:

$$p(\mathbf{w} | \mathbf{y}) = \prod_{m=1}^M p(w_m | y_m). \quad [7.32]$$

3496 Second, each tag y_m depends only on its predecessor,

$$p(\mathbf{y}) = \prod_{m=1}^M p(y_m | y_{m-1}), \quad [7.33]$$

3497 where $y_0 = \diamond$ in all cases. Due to this **Markov assumption**, probabilistic sequence labeling
 3498 models are known as **hidden Markov models** (HMMs).

3499 The generative process for the hidden Markov model is shown in Algorithm 12. Given
 3500 the parameters $\boldsymbol{\lambda}$ and $\boldsymbol{\phi}$, we can compute $p(\mathbf{w}, \mathbf{y})$ for any token sequence \mathbf{w} and tag se-
 3501 quence \mathbf{y} . The HMM is often represented as a **graphical model** (Wainwright and Jordan,
 3502 2008), as shown in Figure 7.2. This representation makes the independence assumptions
 3503 explicit: if a variable v_1 is probabilistically conditioned on another variable v_2 , then there
 3504 is an arrow $v_2 \rightarrow v_1$ in the diagram. If there are no arrows between v_1 and v_2 , they
 3505 are **conditionally independent**, given each variable's **Markov blanket**. In the hidden
 3506 Markov model, the Markov blanket for each tag y_m includes the “parent” y_{m-1} , and the
 3507 “children” y_{m+1} and w_m .³

3508 It is important to reflect on the implications of the HMM independence assumptions.
 3509 A non-adjacent pair of tags y_m and y_n are conditionally independent; if $m < n$ and we
 3510 are given y_{n-1} , then y_m offers no additional information about y_n . However, if we are
 3511 not given any information about the tags in a sequence, then all tags are probabilistically
 3512 coupled.

³In general graphical models, a variable's Markov blanket includes its parents, children, and its children's other parents (Murphy, 2012).

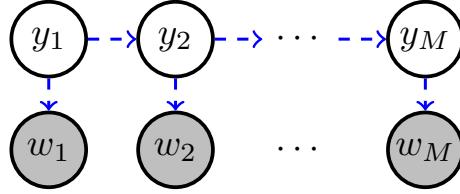


Figure 7.2: Graphical representation of the hidden Markov model. Arrows indicate probabilistic dependencies.

3513 7.4.1 Estimation

3514 The hidden Markov model has two groups of parameters:

3515 **Emission probabilities.** The probability $p_e(w_m | y_m; \phi)$ is the emission probability, since
3516 the words are treated as probabilistically “emitted”, conditioned on the tags.

3517 **Transition probabilities.** The probability $p_t(y_m | y_{m-1}; \lambda)$ is the transition probability,
3518 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,

$$\begin{aligned}\phi_{k,i} &\triangleq \Pr(W_m = i | Y_m = k) = \frac{\text{count}(W_m = i, Y_m = k)}{\text{count}(Y_m = k)} \\ \lambda_{k,k'} &\triangleq \Pr(Y_m = k' | Y_{m-1} = k) = \frac{\text{count}(Y_m = k', Y_{m-1} = k)}{\text{count}(Y_{m-1} = k)}.\end{aligned}$$

3519 Smoothing is more important for the emission probability than the transition probability,
3520 because the vocabulary is much larger than the number of tags.

3521 7.4.2 Inference

3522 The goal of inference in the hidden Markov model is to find the highest probability tag
3523 sequence,

$$\hat{y} = \underset{y}{\operatorname{argmax}} p(y | w). \quad [7.34]$$

3524 As in Naïve Bayes, it is equivalent to find the tag sequence with the highest *log*-probability,
3525 since the logarithm is a monotonically increasing function. It is furthermore equivalent
3526 to maximize the joint probability $p(y, w) = p(y | w) \times p(w) \propto p(y | w)$, which is pro-
3527 portional to the conditional probability. Putting these observations together, the inference

problem can be reformulated as,

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y}} \log p(\mathbf{y}, \mathbf{w}). \quad [7.35]$$

We can now apply the HMM independence assumptions:

$$\log p(\mathbf{y}, \mathbf{w}) = \log p(\mathbf{y}) + \log p(\mathbf{w} \mid \mathbf{y}) \quad [7.36]$$

$$= \sum_{m=1}^{M+1} \log p_Y(y_m \mid y_{m-1}) + \log p_{W|Y}(w_m \mid y_m) \quad [7.37]$$

$$= \sum_{m=1}^{M+1} \log \lambda_{y_m, y_{m-1}} + \log \phi_{y_m, w_m} \quad [7.38]$$

$$= \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}), \quad [7.39]$$

where,

$$s_m(y_m, y_{m-1}) \triangleq \log \lambda_{y_m, y_{m-1}} + \log \phi_{y_m, w_m}, \quad [7.40]$$

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,

$$v_m(y_m) = \max_{\mathbf{y}_{1:m-1}} p(\mathbf{y}_{1:m-1}, y_m, \mathbf{w}_{1:m}) \quad [7.50]$$

$$= \max_{y_{m-1}} p(y_m, w_m | y_{m-1}) \times \max_{\mathbf{y}_{1:m-2}} p(\mathbf{y}_{1:m-2}, y_{m-1}, \mathbf{w}_{1:m-1}) \quad [7.51]$$

$$= \max_{y_{m-1}} p(y_m, w_m | y_{m-1}) \times v_{m-1}(y_{m-1}) \quad [7.52]$$

$$= p_{w|y}(w_m | y_m) \times \max_{y_{m-1}} p_y(y_m | y_{m-1}) \times v_{m-1}(y_{m-1}). \quad [7.53]$$

3531 Each Viterbi variable is computed by *maximizing* over a set of *products*. Thus, the Viterbi
 3532 algorithm is a special case of the **max-product algorithm** for inference in graphical mod-
 3533 els (Wainwright and Jordan, 2008). However, the product of probabilities tends towards
 3534 zero over long sequences, so the log-probability version of Viterbi is recommended in
 3535 practical implementations.

3536 7.5 Discriminative sequence labeling with features

3537 Today, hidden Markov models are rarely used for supervised sequence labeling. This is
 3538 because HMMs are limited to only two phenomena:

- 3539 • word-tag compatibility, via the emission probability $p_{W|Y}(w_m | y_m)$;
- 3540 • local context, via the transition probability $p_Y(y_m | y_{m-1})$.

3541 The Viterbi algorithm permits the inclusion of richer information in the local scoring func-
 3542 tion $\psi(\mathbf{w}_{1:M}, y_m, y_{m-1}, m)$, which can be defined as a weighted sum of arbitrary local *fea-*
 3543 *tures*,

$$\psi(\mathbf{w}, y_m, y_{m-1}, m) = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m), \quad [7.54]$$

3544 where \mathbf{f} is a locally-defined feature function, and $\boldsymbol{\theta}$ is a vector of weights.

The local decomposition of the scoring function Ψ is reflected in a corresponding decomposition of the feature function:

$$\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.55]$$

$$= \sum_{m=1}^{M+1} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.56]$$

$$= \boldsymbol{\theta} \cdot \sum_{m=1}^{M+1} \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.57]$$

$$= \boldsymbol{\theta} \cdot \mathbf{f}^{(\text{global})}(\mathbf{w}, \mathbf{y}_{1:M}), \quad [7.58]$$

3545 where $\mathbf{f}^{(\text{global})}(\mathbf{w}, \mathbf{y})$ is a global feature vector, which is a sum of local feature vectors,

$$\mathbf{f}^{(\text{global})}(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \mathbf{f}(\mathbf{w}_{1:M}, y_m, y_{m-1}, m), \quad [7.59]$$

3546 with $y_{M+1} = \diamond$ and $y_0 = \diamond$ by construction.

3547 Let's now consider what additional information these features might encode.

3548 **Word affix features.** Consider the problem of part-of-speech tagging on the first four
3549 lines of the poem *Jabberwocky* (Carroll, 1917):

3550 (7.3) 'Twas brillig, and the slithy toves
3551 Did gyre and gimble in the wabe:
3552 All mimsy were the borogoves,
3553 And the mome raths outgrabe.

3554 Many of these words were made up by the author of the poem, so a corpus would offer
3555 no information about their probabilities of being associated with any particular part of
3556 speech. Yet it is not so hard to see what their grammatical roles might be in this passage.
3557 Context helps: for example, the word *slithy* follows the determiner *the*, so it is probably a
3558 noun or adjective. Which do you think is more likely? The suffix *-thy* is found in a number
3559 of adjectives, like *frothy*, *healthy*, *pithy*, *worthy*. It is also found in a handful of nouns — e.g.,
3560 *apathy*, *sympathy* — but nearly all of these have the longer coda *-pathy*, unlike *slithy*. So the
3561 suffix gives some evidence that *slithy* is an adjective, and indeed it is: later in the text we
3562 find that it is a combination of the adjectives *lithe* and *slimy*.⁴

⁴Morphology is the study of how words are formed from smaller linguistic units. Computational approaches to morphological analysis are touched on in chapter 9; Bender (2013) provides a good overview of the underlying linguistic principles.

3563 **Fine-grained context.** The hidden Markov model captures contextual information in the
 3564 form of part-of-speech tag bigrams. But sometimes, the necessary contextual information
 3565 is more specific. Consider the noun phrases *this fish* and *these fish*. Many part-of-speech
 3566 tagsets distinguish between singular and plural nouns, but do not distinguish between
 3567 singular and plural determiners.⁵ A hidden Markov model would be unable to correctly
 3568 label *fish* as singular or plural in both of these cases, because it only has access to two
 3569 features: the preceding tag (determiner in both cases) and the word (*fish* in both cases).
 3570 The classification-based tagger discussed in § 7.1 had the ability to use preceding and suc-
 3571 ceeding words as features, and it can also be incorporated into a Viterbi-based sequence
 3572 labeler as a local feature.

Example Consider the tagging D J N (determiner, adjective, noun) for the sequence *the slithy toves*, so that

$$\mathbf{w} = \text{the slithy toves}$$

$$\mathbf{y} = \text{D J N}.$$

Let's create the feature vector for this example, assuming that we have word-tag features (indicated by W), tag-tag features (indicated by T), and suffix features (indicated by M). You can assume that you have access to a method for extracting the suffix *-thy* from *slithy*, *-es* from *toves*, and \emptyset from *the*, indicating that this word has no suffix.⁶ The resulting feature vector is,

$$\begin{aligned} f(\text{the slithy toves, D J N}) &= f(\text{the slithy toves, D}, \diamond, 1) \\ &\quad + f(\text{the slithy toves, J}, D, 2) \\ &\quad + f(\text{the slithy toves, N}, J, 3) \\ &\quad + f(\text{the slithy toves, } \blacklozenge, N, 4) \\ &= \{(T : \diamond, D), (W : \text{the}, D), (M : \emptyset, D), \\ &\quad (T : D, J), (W : \text{slithy}, J), (M : -thy, J), \\ &\quad (T : J, N), (W : \text{toves}, N), (M : -es, N) \\ &\quad (T : N, \blacklozenge)\}. \end{aligned}$$

3573 These examples show that local features can incorporate information that lies beyond
 3574 the scope of a hidden Markov model. Because the features are local, it is possible to apply
 3575 the Viterbi algorithm to identify the optimal sequence of tags. The remaining question

⁵For example, the Penn Treebank tagset follows these conventions.

⁶Such a system is called a **morphological segmenter**. The task of morphological segmentation is briefly described in § 9.1.4.4; a well known segmenter is Morfessor (Creutz and Lagus, 2007). In real applications, a typical approach is to include features for all orthographic suffixes up to some maximum number of characters: for *slithy*, we would have suffix features for *-y*, *-hy*, and *-thy*.

3576 is how to estimate the weights on these features. § 2.2 presented three main types of
 3577 discriminative classifiers: perceptron, support vector machine, and logistic regression.
 3578 Each of these classifiers has a structured equivalent, enabling it to be trained from labeled
 3579 sequences rather than individual tokens.

3580 **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,

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} \theta \cdot f(\mathbf{w}, \mathbf{y}) \quad [7.62]$$

$$\theta^{(t+1)} \leftarrow \theta^{(t)} + f(\mathbf{w}, \mathbf{y}) - f(\mathbf{w}, \hat{\mathbf{y}}). \quad [7.63]$$

3581 This learning algorithm is called **structured perceptron**, because it learns to predict the
 3582 structured output \mathbf{y} . The only difference is that instead of computing \hat{y} by enumerating
 3583 the entire set \mathcal{Y} , the Viterbi algorithm is used to efficiently search the set of possible tag-
 3584 gings, \mathcal{Y}^M . Structured perceptron can be applied to other structured outputs as long as
 3585 efficient inference is possible. As in perceptron classification, weight averaging is crucial
 3586 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:

$$\theta_{(fish,V)} \leftarrow \theta_{(fish,V)} + 1, \quad \theta_{(fish,N)} \leftarrow \theta_{(fish,N)} - 1 \quad [7.64]$$

$$\theta_{(V,V)} \leftarrow \theta_{(V,V)} + 1, \quad \theta_{(V,N)} \leftarrow \theta_{(V,N)} - 1 \quad [7.65]$$

$$\theta_{(V,\blacklozenge)} \leftarrow \theta_{(V,\blacklozenge)} + 1, \quad \theta_{(N,\blacklozenge)} \leftarrow \theta_{(N,\blacklozenge)} - 1. \quad [7.66]$$

3587 **7.5.2 Structured support vector machines**

3588 Large-margin classifiers such as the support vector machine improve on the perceptron by
 3589 pushing the classification boundary away from the training instances. The same idea can

be applied to sequence labeling. A support vector machine in which the output is a structured object, such as a sequence, is called a **structured support vector machine** (Tsochan-taridis et al., 2004).⁷

In classification, we formalized the large-margin constraint as,

$$\forall \mathbf{y} \neq \mathbf{y}^{(i)}, \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}^{(i)}) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}) \geq 1, \quad [7.67]$$

requiring a margin of at least 1 between the scores for all labels \mathbf{y} that are not equal to the correct label $\mathbf{y}^{(i)}$. The weights $\boldsymbol{\theta}$ are then learned by constrained optimization (see § 2.3.2).

This idea can be applied to sequence labeling by formulating an equivalent set of constraints for all possible labelings $\mathcal{Y}(\mathbf{w})$ for an input \mathbf{w} . However, there are two problems. First, in sequence labeling, some predictions are more wrong than others: we may miss only one tag out of fifty, or we may get all fifty wrong. We would like our learning algorithm to be sensitive to this difference. Second, the number of constraints is equal to the number of possible labelings, which is exponentially large in the length of the sequence.

The first problem can be addressed by adjusting the constraint to require larger margins for more serious errors. Let $c(\mathbf{y}^{(i)}, \hat{\mathbf{y}}) \geq 0$ represent the *cost* of predicting label $\hat{\mathbf{y}}$ when the true label is $\mathbf{y}^{(i)}$. We can then generalize the margin constraint,

$$\forall \mathbf{y}, \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) \geq c(\mathbf{y}^{(i)}, \mathbf{y}). \quad [7.68]$$

This cost-augmented margin constraint specializes to the constraint in Equation 7.67 if we choose the delta function $c(\mathbf{y}^{(i)}, \mathbf{y}) = \delta((\mathbf{y}^{(i)} \neq \mathbf{y}))$. A more expressive cost function is the **Hamming cost**,

$$c(\mathbf{y}^{(i)}, \mathbf{y}) = \sum_{m=1}^M \delta(y_m^{(i)} \neq y_m), \quad [7.69]$$

which computes the number of errors in \mathbf{y} . By incorporating the cost function as the margin constraint, we require that the true labeling be separated from the alternatives by a margin that is proportional to the number of incorrect tags in each alternative labeling.

The second problem is that the number of constraints is exponential in the length of the sequence. This can be addressed by focusing on the prediction $\hat{\mathbf{y}}$ that *maximally* violates the margin constraint. This prediction can be identified by solving the following **cost-augmented decoding** problem:

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \neq \mathbf{y}^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) + c(\mathbf{y}^{(i)}, \mathbf{y}) \quad [7.70]$$

$$= \operatorname{argmax}_{\mathbf{y} \neq \mathbf{y}^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) + c(\mathbf{y}^{(i)}, \mathbf{y}), \quad [7.71]$$

⁷This model is also known as a **max-margin Markov network** (Taskar et al., 2003), emphasizing that the scoring function is constructed from a sum of components, which are Markov independent.

3611 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,

$$\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) - \max_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} (\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}) + c(\mathbf{y}^{(i)}, \mathbf{y})) \geq 0. \quad [7.72]$$

3612 If the score for $\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})$ is greater than the cost-augmented score for all alternatives,
 3613 then the constraint will be met. The name “cost-augmented decoding” is due to the fact
 3614 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
 3615 an additional term for the cost. Essentially, we want to train against predictions that are
 3616 strong and wrong: they should score highly according to the model, yet incur a large loss
 3617 with respect to the ground truth. Training adjusts the weights to reduce the score of these
 3618 predictions.

3619 For cost-augmented decoding to be tractable, the cost function must decompose into
 3620 local parts, just as the feature function $f(\cdot)$ does. The Hamming cost, defined above,
 3621 obeys this property. To perform cost-augmented decoding using the Hamming cost, we
 3622 need only to add features $f_m(y_m) = \delta(y_m \neq y_m^{(i)})$, and assign a constant weight of 1 to
 3623 these features. Decoding can then be performed using the Viterbi algorithm.⁸

As with large-margin classifiers, it is possible to formulate the learning problem in an unconstrained form, by combining a regularization term on the weights and a Lagrangian for the constraints:

$$\min_{\theta} \frac{1}{2} \|\theta\|_2^2 - C \left(\sum_i \theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) - \max_{\mathbf{y} \in \mathcal{Y}(\mathbf{w}^{(i)})} [\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}) + c(\mathbf{y}^{(i)}, \mathbf{y})] \right), \quad [7.73]$$

3624 In this formulation, C is a parameter that controls the tradeoff between the regularization
 3625 term and the margin constraints. A number of optimization algorithms have been
 3626 proposed for structured support vector machines, some of which are discussed in § 2.3.2.
 3627 An empirical comparison by Kummerfeld et al. (2015) shows that stochastic subgradient
 3628 descent — which is essentially a cost-augmented version of the structured perceptron —
 3629 is highly competitive.

3630 7.5.3 Conditional random fields

3631 The **conditional random field** (CRF; Lafferty et al., 2001) is a conditional probabilistic
 3632 model for sequence labeling; just as structured perceptron is built on the perceptron clas-
 3633 sifier, conditional random fields are built on the logistic regression classifier.⁹ The basic

⁸Are there cost functions that do not decompose into local parts? Suppose we want to assign a constant loss c to any prediction $\hat{\mathbf{y}}$ in which k or more predicted tags are incorrect, and zero loss otherwise. This loss function is combinatorial over the predictions, and thus we cannot decompose it into parts.

⁹The name “Conditional Random Field” is derived from **Markov random fields**, a general class of models in which the probability of a configuration of variables is proportional to a product of scores across pairs (or

3634 probability model is,

$$p(\mathbf{y} \mid \mathbf{w}) = \frac{\exp(\Psi(\mathbf{w}, \mathbf{y}))}{\sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp(\Psi(\mathbf{w}, \mathbf{y}'))}. \quad [7.74]$$

3635 This is almost identical to logistic regression, but because the label space is now tag
 3636 sequences, we require efficient algorithms for both **decoding** (searching for the best tag
 3637 sequence given a sequence of words \mathbf{w} and a model θ) and for **normalizing** (summing
 3638 over all tag sequences). These algorithms will be based on the usual locality assumption
 3639 on the scoring function, $\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m)$.

3640 **7.5.3.1 Decoding in CRFs**

Decoding — finding the tag sequence $\hat{\mathbf{y}}$ that maximizes $p(\mathbf{y} \mid \mathbf{w})$ — is a direct application of the Viterbi algorithm. The key observation is that the decoding problem does not depend on the denominator of $p(\mathbf{y} \mid \mathbf{w})$,

$$\begin{aligned} \hat{\mathbf{y}} &= \operatorname{argmax}_{\mathbf{y}} \log p(\mathbf{y} \mid \mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{y}} \Psi(\mathbf{y}, \mathbf{w}) - \log \sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp \Psi(\mathbf{y}', \mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{y}} \Psi(\mathbf{y}, \mathbf{w}) = \operatorname{argmax}_{\mathbf{y}} \sum_{m=1}^{M+1} s(y_m, y_{m-1}). \end{aligned}$$

3641 This is identical to the decoding problem for structured perceptron, so the same Viterbi
 3642 recurrence as defined in Equation 7.22 can be used.

3643 **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.

3644 where λ controls the amount of regularization. The final term in Equation 7.76 is a sum
 3645 over all possible labelings. This term is the log of the denominator in Equation 7.74, some-
 3646 times known as the **partition function**.¹⁰ There are $|\mathcal{Y}|^M$ possible labelings of an input of
 3647 size M , so we must again exploit the decomposition of the scoring function to compute
 3648 this sum efficiently.

The sum $\sum_{\mathbf{y} \in \mathcal{Y}^{w(i)}} \exp \Psi(\mathbf{y}, \mathbf{w})$ can be computed efficiently using the **forward recurrence**, which is closely related to the Viterbi recurrence. We first define a set of **forward variables**, $\alpha_m(y_m)$, which is equal to the sum of the scores of all paths leading to tag y_m at position m :

$$\alpha_m(y_m) \triangleq \sum_{\mathbf{y}_{1:m-1}} \exp \sum_{n=1}^m s_n(y_n, y_{n-1}) \quad [7.77]$$

$$= \sum_{\mathbf{y}_{1:m-1}} \prod_{n=1}^m \exp s_n(y_n, y_{n-1}). \quad [7.78]$$

Note the similarity to the definition of the Viterbi variable, $v_m(y_m) = \max_{\mathbf{y}_{1:m-1}} \sum_{n=1}^m s_n(y_n, y_{n-1})$. In the hidden Markov model, the Viterbi recurrence had an alternative interpretation as the max-product algorithm (see Equation 7.53); analogously, the forward recurrence is known as the **sum-product algorithm**, because of the form of [7.78]. The forward variable can also be computed through a recurrence:

$$\alpha_m(y_m) = \sum_{\mathbf{y}_{1:m-1}} \prod_{n=1}^m \exp s_n(y_n, y_{n-1}) \quad [7.79]$$

$$= \sum_{y_{m-1}} (\exp s_m(y_m, y_{m-1})) \sum_{\mathbf{y}_{1:m-2}} \prod_{n=1}^{m-1} \exp s_n(y_n, y_{n-1}) \quad [7.80]$$

$$= \sum_{y_{m-1}} (\exp s_m(y_m, y_{m-1})) \times \alpha_{m-1}(y_{m-1}). \quad [7.81]$$

Using the forward recurrence, it is possible to compute the denominator of the conditional probability,

$$\sum_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} \Psi(\mathbf{w}, \mathbf{y}) = \sum_{\mathbf{y}_{1:M}} s_{M+1}(\blacklozenge, y_M) \prod_{m=1}^M s_m(y_m, y_{m-1}) \quad [7.82]$$

$$= \alpha_{M+1}(\blacklozenge). \quad [7.83]$$

¹⁰The terminology of “potentials” and “partition functions” comes from statistical mechanics (Bishop, 2006).

The conditional log-likelihood can be rewritten,

$$\ell = \frac{\lambda}{2} \|\boldsymbol{\theta}\|^2 - \sum_{i=1}^N \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) + \log \alpha_{M+1}(\blacklozenge). \quad [7.84]$$

- 3649 Probabilistic programming environments, such as `Torch` (Collobert et al., 2011) and `dynet` (Neu-
 3650 big et al., 2017), can compute the gradient of this objective using automatic differentiation.
 3651 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:

$$\frac{d\ell}{d\theta_j} = \lambda \theta_j + \sum_{i=1}^N E[f_j(\mathbf{w}^{(i)}, \mathbf{y})] - f_j(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}), \quad [7.85]$$

- 3652 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-
 3653 quence $\mathbf{y}^{(i)}$. The expected feature counts are computed “under the hood” when automatic
 3654 differentiation is applied to Equation 7.84 (Eisner, 2016).

- 3655 Before the widespread use of automatic differentiation, it was common to compute
 3656 the feature expectations from marginal tag probabilities $p(y_m | \mathbf{w})$. These marginal prob-
 3657 abilities are sometimes useful on their own, and can be computed using the **forward-**
 3658 **backward algorithm**. This algorithm combines the forward recurrence with an equivalent
 3659 **backward recurrence**, which traverses the input from w_M back to w_1 .

3660 7.5.3.3 *Forward-backward algorithm

Marginal probabilities over tag bigrams can be written as,¹¹

$$\Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}) = \frac{\sum_{\mathbf{y}: Y_m=k, Y_{m-1}=k'} \prod_{n=1}^M \exp s_n(y_n, y_{n-1})}{\sum_{\mathbf{y}'} \prod_{n=1}^M \exp s_n(y'_n, y'_{n-1})}. \quad [7.86]$$

The numerator sums over all tag sequences that include the transition $(Y_{m-1} = k') \rightarrow (Y_m = k)$. Because we are only interested in sequences that include the tag bigram, this sum can be decomposed into three parts: the *prefixes* $\mathbf{y}_{1:m-1}$, terminating in $Y_{m-1} = k'$; the

¹¹Recall the notational convention of upper-case letters for random variables, e.g. Y_m , and lower case letters for specific values, e.g., y_m , so that $Y_m = k$ is interpreted as the event of random variable Y_m taking the value k .

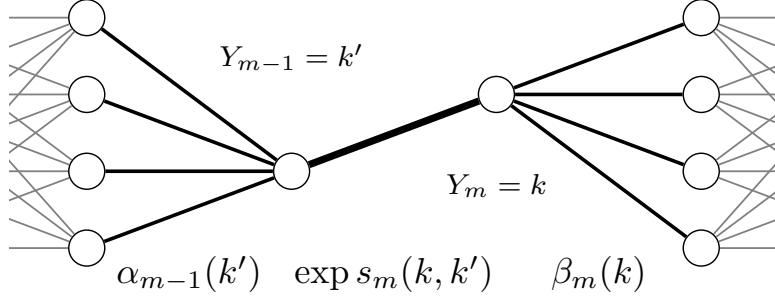


Figure 7.3: A schematic illustration of the computation of the marginal probability $\Pr(Y_{m-1} = k', Y_m = k)$, using the forward score $\alpha_{m-1}(k')$ and the backward score $\beta_m(k)$.

transition $(Y_{m-1} = k') \rightarrow (Y_m = k)$; and the *suffixes* $\mathbf{y}_{m:M}$, beginning with the tag $Y_m = k$:

$$\sum_{\mathbf{y}: Y_m = k, Y_{m-1} = k'} \prod_{n=1}^M \exp s_n(y_n, y_{n-1}) = \sum_{\mathbf{y}_{1:m-1}: Y_{m-1} = k'} \prod_{n=1}^{m-1} \exp s_n(y_n, y_{n-1}) \times \exp s_m(k, k') \times \sum_{\mathbf{y}_{m:M}: Y_m = k} \prod_{n=m+1}^{M+1} \exp s_n(y_n, y_{n-1}). \quad [7.87]$$

The result is product of three terms: a score that sums over all the ways to get to the position $(Y_{m-1} = k')$, a score for the transition from k' to k , and a score that sums over all the ways of finishing the sequence from $(Y_m = k)$. The first term of Equation 7.87 is equal to the **forward variable**, $\alpha_{m-1}(k')$. The third term — the sum over ways to finish the sequence — can also be defined recursively, this time moving over the trellis from right to left, which is known as the **backward recurrence**:

$$\beta_m(k) \triangleq \sum_{\mathbf{y}_{m:M}: Y_m = k} \prod_{n=m}^{M+1} \exp s_n(y_n, y_{n-1}) \quad [7.88]$$

$$= \sum_{k' \in \mathcal{Y}} \exp s_{m+1}(k', k) \sum_{\mathbf{y}_{m+1:M}: Y_m = k'} \prod_{n=m+1}^{M+1} \exp s_n(y_n, y_{n-1}) \quad [7.89]$$

$$= \sum_{k' \in \mathcal{Y}} \exp s_{m+1}(k', k) \times \beta_{m+1}(k'). \quad [7.90]$$

³⁶⁶¹ 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 :

$$\psi_m(y) = \beta_y \cdot \mathbf{h}_m \quad [7.93]$$

$$\hat{y}_m = \operatorname{argmax}_y \psi_m(y). \quad [7.94]$$

The score $\psi_m(y)$ can also be converted into a probability distribution using the usual softmax operation,

$$p(y | \mathbf{w}_{1:m}) = \frac{\exp \psi_m(y)}{\sum_{y' \in \mathcal{Y}} \exp \psi_m(y')}. \quad [7.95]$$

3680 Using this transformation, it is possible to train the tagger from the negative log-likelihood
 3681 of the tags, as in a conditional random field. Alternatively, a hinge loss or margin loss
 3682 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:

$$\overleftarrow{\mathbf{h}}_m = g(\mathbf{x}_m, \overleftarrow{\mathbf{h}}_{m+1}), \quad m = 1, 2, \dots, M. \quad [7.96]$$

3683 The hidden states of the left-to-right RNN are denoted $\overrightarrow{\mathbf{h}}_m$. The left-to-right and right-to-
 3684 left vectors are concatenated, $\mathbf{h}_m = [\overleftarrow{\mathbf{h}}_m; \overrightarrow{\mathbf{h}}_m]$. The scoring function in Equation 7.93 is
 3685 applied to this concatenated vector.

3686 Bidirectional RNN tagging has several attractive properties. Ideally, the representa-
 3687 tion \mathbf{h}_m summarizes the useful information from the surrounding context, so that it is not
 3688 necessary to design explicit features to capture this information. If the vector \mathbf{h}_m is an ad-
 3689 equate summary of this context, then it may not even be necessary to perform the tagging
 3690 jointly: in general, the gains offered by joint tagging of the entire sequence are diminished
 3691 as the individual tagging model becomes more powerful. Using backpropagation, the
 3692 word vectors \mathbf{x} can be trained “end-to-end”, so that they capture word properties that are
 3693 useful for the tagging task. Alternatively, if limited labeled data is available, we can use
 3694 word embeddings that are “pre-trained” from unlabeled data, using a language modeling
 3695 objective (as in § 6.3) or a related word embedding technique (see chapter 14). It is even
 3696 possible to combine both fine-tuned and pre-trained embeddings in a single model.

3697 **Neural structure prediction** The bidirectional recurrent neural network incorporates in-
 3698 formation from throughout the input, but each tagging decision is made independently.
 3699 In some sequence labeling applications, there are very strong dependencies between tags:
 3700 it may even be impossible for one tag to follow another. In such scenarios, the tagging
 3701 decision must be made jointly across the entire sequence.

3702 Neural sequence labeling can be combined with the Viterbi algorithm by defining the
 3703 local scores as:

$$s_m(y_m, y_{m-1}) = \beta_{y_m} \cdot \mathbf{h}_m + \eta_{y_{m-1}, y_m}, \quad [7.97]$$

3704 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}
 3705 is a scalar parameter for the tag transition (y_{m-1}, y_m) . These local scores can then be
 3706 incorporated into the Viterbi algorithm for inference, and into the forward algorithm for
 3707 training. This model is shown in Figure 7.4. It can be trained from the conditional log-
 3708 likelihood objective defined in Equation 7.76, backpropagating to the tagging parameters

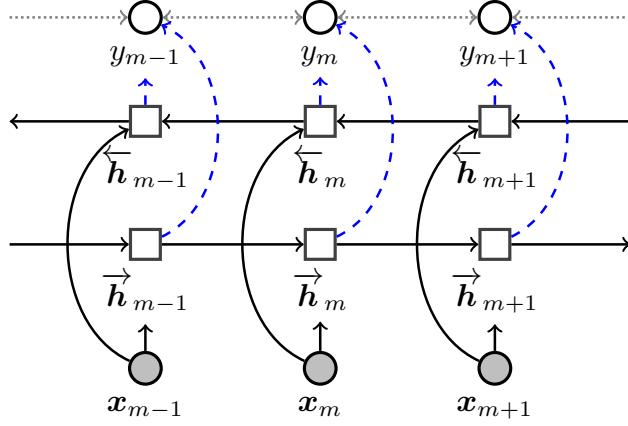


Figure 7.4: Bidirectional LSTM for sequence labeling. The solid lines indicate computation, the dashed lines indicate probabilistic dependency, and the dotted lines indicate the optional additional probabilistic dependencies between labels in the biLSTM-CRF.

3709 β and η , as well as the parameters of the RNN. This model is called the **LSTM-CRF**, due
 3710 to its combination of aspects of the long short-term memory and conditional random field
 3711 models (Huang et al., 2015).

3712 The LSTM-CRF is especially effective on the task of **named entity recognition** (Lample
 3713 et al., 2016), a sequence labeling task that is described in detail in § 8.3. This task has strong
 3714 dependencies between adjacent tags, so structure prediction is especially important.

3715 7.6.2 Character-level models

3716 As in language modeling, rare and unseen words are a challenge: if we encounter a word
 3717 that was not in the training data, then there is no obvious choice for the word embed-
 3718 ding x_m . One solution is to use a generic **unseen word** embedding for all such words.
 3719 However, in many cases, properties of unseen words can be guessed from their spellings.
 3720 For example, *whimsical* does not appear in the Universal Dependencies (UD) English Tree-
 3721 bank, yet the suffix *-al* makes it likely to be adjective; by the same logic, *unflinchingly* is
 3722 likely to be an adverb, and *barnacle* is likely to be a noun.

3723 In feature-based models, these morphological properties were handled by suffix fea-
 3724 tures; in a neural network, they can be incorporated by constructing the embeddings of
 3725 unseen words from their spellings or morphology. One way to do this is to incorporate
 3726 an additional layer of bidirectional RNNs, one for each word in the vocabulary (Ling
 3727 et al., 2015). For each such character-RNN, the inputs are the characters, and the output
 3728 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.

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:

$$\Pr(W = i | Y = k) = \phi_{k,i} = \frac{E[\text{count}(W = i, Y = k)]}{E[\text{count}(Y = k)]}$$

$$\Pr(Y_m = k | Y_{m-1} = k') = \lambda_{k',k} = \frac{E[\text{count}(Y_m = k, Y_{m-1} = k')]}{E[\text{count}(Y_{m-1} = k')]}.$$

The expected counts are computed in the E-step, using the forward and backward

(c) Jacob Eisenstein 2018. Work in progress.

recurrences. The local scores follow the usual definition for hidden Markov models,

$$s_m(k, k') = \log p_E(w_m | Y_m = k; \phi) + \log p_T(Y_m = k | Y_{m-1} = k'; \lambda). \quad [7.98]$$

The expected transition counts for a single instance are,

$$E[\text{count}(Y_m = k, Y_{m-1} = k') | \mathbf{w}] = \sum_{m=1}^M \Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}) \quad [7.99]$$

$$= \frac{\sum_{\mathbf{y}: Y_m = k, Y_{m-1} = k'} \prod_{n=1}^M \exp s_n(y_n, y_{n-1})}{\sum_{\mathbf{y}'} \prod_{n=1}^M \exp s_n(y'_n, y'_{n-1})}. \quad [7.100]$$

As described in § 7.5.3.3, these marginal probabilities can be computed from the forward-backward recurrence,

$$\Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}) = \frac{\alpha_{m-1}(k') \times s_m(k, k') \times \beta_m(k)}{\alpha_{M+1}(\diamond)}. \quad [7.101]$$

In a hidden Markov model, each element of the forward-backward computation has a special interpretation:

$$\alpha_{m-1}(k') = p(Y_{m-1} = k', \mathbf{w}_{1:m-1}) \quad [7.102]$$

$$s_m(k, k') = p(Y_m = k, w_m | Y_{m-1} = k') \quad [7.103]$$

$$\beta_m(k) = p(\mathbf{w}_{m+1:M} | Y_m = k). \quad [7.104]$$

Applying the conditional independence assumptions of the hidden Markov model (defined in Algorithm 12), the product is equal to the joint probability of the tag bigram and the entire input,

$$\begin{aligned} \alpha_{m-1}(k') \times s_m(k, k') \times \beta_m(k) &= p(Y_{m-1} = k', \mathbf{w}_{1:m-1}) \\ &\quad \times p(Y_m = k, w_m | Y_{m-1} = k') \\ &\quad \times p(\mathbf{w}_{m+1:M} | Y_m = k) \\ &= p(Y_{m-1} = k', Y_m = k, \mathbf{w}_{1:M}). \end{aligned} \quad [7.105]$$

Dividing by $\alpha_{M+1}(\diamond) = p(\mathbf{w}_{1:M})$ gives the desired probability,

$$\frac{\alpha_{m-1}(k') \times s_m(k, k') \times \beta_m(k)}{\alpha_{M+1}(\diamond)} = \frac{p(Y_{m-1} = k', Y_m = k, \mathbf{w}_{1:M})}{p(\mathbf{w}_{1:M})} \quad [7.106]$$

$$= \Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}_{1:M}). \quad [7.107]$$

The expected emission counts can be computed in a similar manner, using the product $\alpha_m(k) \times \beta_m(k)$.

3757 **7.7.1 Linear dynamical systems**

3758 The forward-backward algorithm can be viewed as Bayesian state estimation in a discrete
 3759 state space. In a continuous state space, $\mathbf{y}_m \in \mathbb{R}^K$, the equivalent algorithm is the **Kalman**
 3760 **smoother**. It also computes marginals $p(\mathbf{y}_m | \mathbf{x}_{1:M})$, using a similar two-step algorithm
 3761 of forward and backward passes. Instead of computing a trellis of values at each step, the
 3762 Kalman smoother computes a probability density function $q_{\mathbf{y}_m}(\mathbf{y}_m; \boldsymbol{\mu}_m, \Sigma_m)$, character-
 3763 ized by a mean $\boldsymbol{\mu}_m$ and a covariance Σ_m around the latent state. Connections between the
 3764 Kalman Smoother and the forward-backward algorithm are elucidated by Minka (1999)
 3765 and Murphy (2012).

3766 **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):

$$p(y_m | \mathbf{y}_{-m}, \mathbf{w}_{1:M}) \propto p(w_m | y_m) p(y_m | \mathbf{y}_{-m}). \quad [7.108]$$

3767 Gibbs Sampling has been applied to unsupervised part-of-speech tagging by Goldwater
 3768 and Griffiths (2007). **Beam sampling** is a more sophisticated sampling algorithm, which
 3769 randomly draws entire sequences $\mathbf{y}_{1:M}$, rather than individual tags y_m ; this algorithm
 3770 was applied to unsupervised part-of-speech tagging by Van Gael et al. (2009). Spectral
 3771 learning (see § 5.5.2) can also be applied to sequence labeling. By factoring matrices of
 3772 co-occurrence counts of word bigrams and trigrams (Song et al., 2010; Hsu et al., 2012), it
 3773 is possible to obtain globally optimal estimates of the transition and emission parameters,
 3774 under mild assumptions.

3775 **7.7.3 Semiring Notation and the Generalized Viterbi Algorithm**

The Viterbi and Forward recurrences can each be performed over probabilities or log probabilities, yielding a total of four closely related recurrences. These four recurrence scan in fact be expressed as a single recurrence in a more general notation, known as **semiring algebra**. Let the symbol \oplus represent generalized addition, and the symbol \otimes represent generalized multiplication.¹² Given these operators, we can denote a general-

¹²In a semiring, the addition and multiplication operators must both obey associativity, and multiplication must distribute across addition; the addition operator must be commutative; there must be additive and multiplicative identities $\bar{0}$ and $\bar{1}$, such that $a \oplus \bar{0} = a$ and $a \otimes \bar{1} = a$; and there must be a multiplicative annihilator $\bar{0}$, such that $a \otimes \bar{0} = \bar{0}$.

ized Viterbi recurrence as,

$$v_m(k) = \bigoplus_{k' \in \mathcal{Y}} s_m(k, k') \otimes v_{m-1}(k'). \quad [7.109]$$

3776 Each recurrence that we have seen so far is a special case of this generalized Viterbi
 3777 recurrence:

- 3778 • In the max-product Viterbi recurrence over probabilities, the \oplus operation corre-
 3779 sponds to maximization, and the \otimes operation corresponds to multiplication.
- 3780 • In the forward recurrence over probabilities, the \oplus operation corresponds to addi-
 3781 tion, and the \otimes operation corresponds to multiplication.
- 3782 • In the max-product Viterbi recurrence over log-probabilities, the \oplus operation corre-
 3783 sponds to maximization, and the \otimes operation corresponds to addition.¹³
- 3784 • In the forward recurrence over log-probabilities, the \oplus operation corresponds to log-
 3785 addition, $a \oplus b = \log(e^a + e^b)$. The \otimes operation corresponds to addition.

3786 The mathematical abstraction offered by semiring notation can be applied to the soft-
 3787 ware implementations of these algorithms, yielding concise and modular implemen-
 3788 tations. The OPENFST library (Allauzen et al., 2007) is an example of a software package in
 3789 which the algorithms are parametrized by the choice of semiring.

3790 Exercises

- 3791 1. Consider the garden path sentence, *The old man the boat*. Given word-tag and tag-tag
 3792 features, what inequality in the weights must hold for the correct tag sequence to
 3793 outscore the garden path tag sequence for this example?
- 3794 2. Sketch out an algorithm for a variant of Viterbi that returns the top- n label se-
 3795 quences. What is the time and space complexity of this algorithm?
- 3796 3. Show how to compute the marginal probability $\Pr(y_{m-2} = k, y_m = k' \mid \mathbf{w}_{1:M})$, in
 3797 terms of the forwards and backward variables, and the potentials $s_n(y_n, y_{n-1})$.
- 3798 4. Suppose you receive a stream of text, where some of tokens have been replaced at
 3799 random with *NOISE*. For example:
 - 3800 • Source: *I try all things, I achieve what I can*
 - 3801 • Message received: *I try NOISE NOISE, I NOISE what I NOISE*

¹³This is sometimes called the **tropical semiring**, in honor of the Brazilian mathematician Imre Simon.

3802 Assume you have access to a pre-trained bigram language model, which gives prob-
3803 abilities $p(w_m \mid w_{m-1})$. These probabilities can be assumed to be non-zero for all
3804 bigrams.

- 3805 a) Show how to use the Viterbi algorithm to try to recover the source by maxi-
3806 mizing the bigram language model log-probability. Specifically, set the scores
3807 $s_m(y_m, y_{m-1})$ so that the Viterbi algorithm selects a sequence of words that
3808 maximizes the bigram language model log-probability, *while leaving the non-*
3809 *noise tokens intact*. Your solution should not modify the logic of the Viterbi
3810 algorithm, it should only set the scores $s_m(y_m, y_{m-1})$.
- 3811 b) An alternative solution is to iterate through the text from $m \in \{1, 2, \dots, M\}$,
3812 replacing each noise token with the word that maximizes $P(w_m \mid w_{m-1})$ ac-
3813 cording to the bigram language model. Given an upper bound on the expected
3814 fraction of tokens for which the two approaches will disagree.
- 3815 5. Consider an RNN tagging model with a tanh activation function on the hidden
3816 layer, and a hinge loss on the output. (The problem also works for the margin loss
3817 and negative log-likelihood.) Suppose you initialize all parameters to zero: this
3818 includes the word embeddings that make up \mathbf{x} , the transition matrix Θ , the out-
3819 put weights β , and the initial hidden state \mathbf{h}_0 . Prove that for any data and for any
3820 gradient-based learning algorithm, all parameters will be stuck at zero.
3821 Extra credit: would a sigmoid activation function avoid this problem?

3822 Chapter 8

3823 Applications of sequence labeling

3824 Sequence labeling has applications throughout natural language processing. This chap-
3825 ter focuses on part-of-speech tagging, morpho-syntactic attribute tagging, named entity
3826 recognition, and tokenization. It also touches briefly on two applications to interactive
3827 settings: dialogue act recognition and the detection of code-switching points between
3828 languages.

3829 8.1 Part-of-speech tagging

3830 The **syntax** of a language is the set of principles under which sequences of words are
3831 judged to be grammatically acceptable by fluent speakers. One of the most basic syntactic
3832 concepts is the **part-of-speech** (POS), which refers to the syntactic role of each word in a
3833 sentence. This concept was used informally in the previous chapter, and you may have
3834 some intuitions from your own study of English. For example, in the sentence *We like*
3835 *vegetarian sandwiches*, you may already know that *we* and *sandwiches* are nouns, *like* is a
3836 verb, and *vegetarian* is an adjective. These labels depend on the context in which the word
3837 appears: in *she eats like a vegetarian*, the word *like* is a preposition, and the word *vegetarian*
3838 is a noun.

3839 Parts-of-speech can help to disentangle or explain various linguistic problems. Recall
3840 Chomsky's proposed distinction in chapter 6:

- 3841 (8.1) Colorless green ideas sleep furiously.
- 3842 (8.2) *Ideas colorless furiously green sleep.

3843 One difference between these two examples is that the first contains part-of-speech transitions
3844 that are typical in English: adjective to adjective, adjective to noun, noun to verb, and verb
3845 to adverb. The second example contains transitions that are unusual: noun to adjective
3846 and adjective to verb. The ambiguity in a headline like,

3847 (8.3) Teacher Strikes Idle Children

3848 can also be explained in terms of parts of speech: in the interpretation that was likely
 3849 intended, *strikes* is a noun and *idle* is a verb; in the alternative explanation, *strikes* is a verb
 3850 and *idle* is an adjective.

3851 Part-of-speech tagging is often taken as a early step in a natural language processing
 3852 pipeline. Indeed, parts-of-speech provide features that can be useful for many of the
 3853 tasks that we will encounter later, such as parsing, coreference resolution, and relation
 3854 extraction.

3855 **8.1.1 Parts-of-Speech**

3856 The **Universal Dependencies** project (UD) is an effort to create syntactically-annotated
 3857 corpora across many languages, using a single annotation standard (Nivre et al., 2016). As
 3858 part of this effort, they have designed a part-of-speech **tagset**, which is meant to capture
 3859 word classes across as many languages as possible.¹ This section describes that inventory,
 3860 giving rough definitions for each of tags, along with supporting examples.

3861 Part-of-speech tags are **morphosyntactic**, rather than **semantic**, categories. This means
 3862 that they describe words in terms of how they pattern together and how they are inter-
 3863 nally constructed (e.g., what suffixes and prefixes they include). For example, you may
 3864 think of a noun as referring to objects or concepts, and verbs as referring to actions or
 3865 events. But events can also be nouns:

3866 (8.4) ... the **howling** of the **shrieking** storm.

3867 Here *howling* and *shrieking* are events, but grammatically they act as a noun and adjective
 3868 respectively.

3869 **8.1.1.1 The Universal Dependency part-of-speech tagset**

3870 The UD tagset is broken up into three groups: open class tags, closed class tags, and
 3871 “others.”

3872 **Open class tags** Nearly all languages contain nouns, verbs, adjectives, and adverbs.²
 3873 These are all **open word classes**, because new words can easily be added to them. The
 3874 UD tagset includes two other tags that are open classes: proper nouns and interjections.

3875 • **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).

3876 (8.5) **Toes** are scarce among veteran **blubber men**.

3877 In English, nouns tend to follow determiners and adjectives, and can play the subject
3878 role in the sentence. They can be marked for the plural number by an -s suffix.

3879 • **Proper nouns** (PROPN) are tokens in names, which uniquely specify a given entity,

3880 (8.6) “**Moby Dick?**” shouted **Ahab**.

3881 • **Verbs** (VERB), according to the UD guidelines, “typically signal events and ac-
3882 tions.” But they are also defined grammatically: they “can constitute a minimal
3883 predicate in a clause, and govern the number and types of other constituents which
3884 may occur in a clause.”³

3885 (8.7) “**Moby Dick?**” shouted Ahab.

3886 (8.8) Shall we **keep chasing** this murderous fish?

3887 English verbs tend to come in between the subject and some number of direct ob-
3888 jects, depending on the verb. They can be marked for **tense** and **aspect** using suffixes
3889 such as *-ed* and *-ing*. (These suffixes are an example of **inflectional morphology**,
3890 which is discussed in more detail in § 9.1.4.)

3891 • **Adjectives** (ADJ) describe properties of entities,

3892 (8.9) Shall we keep chasing this **murderous** fish?

3893 (8.10) Toes are **scarce** among **veteran** blubber men.

3894 In the second example, *scarce* is a predicative adjective, linked to the subject by the
3895 **copula verb** *are*. This means that In contrast, *murderous* and *veteran* are attribute
3896 adjectives, modifying the noun phrase in which they are embedded.

3897 • **Adverbs** (ADV) describe properties of events, and may also modify adjectives or
3898 other adverbs:

3899 (8.11) It is not down on any map; true places **never** are.

3900 (8.12) ... **treacherously** hidden beneath the loveliest tints of azure

3901 (8.13) Not drowned **entirely**, though.

3902 • **Interjections** (INTJ) are used in exclamations, e.g.,

3903 (8.14) **Aye aye!** it was that accursed white whale that razed me.

³<http://universaldependencies.org/u/pos/VERB.html>

3904 **Closed class tags** Closed word classes rarely receive new members. They are sometimes
 3905 referred to as **function words** — as opposed to **content words** — as they have little lexical
 3906 meaning of their own, but rather, help to organize the components of the sentence.

- 3907 • **Adpositions** (ADP) describe the relationship between a complement (usually a noun
 3908 phrase) and another unit in the sentence, typically a noun or verb phrase.

- 3909 (8.15) Toes are scarce **among** veteran blubber men.
 3910 (8.16) It is not **down on** any map.
 3911 (8.17) Give not thyself **up** then.

3912 As the examples show, English generally uses prepositions, which are adpositions
 3913 that appear before their complement. (An exception is *ago*, as in, *we met three days*
 3914 *ago*). Postpositions are used in other languages, such as Japanese and Turkish.

- 3915 • **Auxiliary verbs** (AUX) are a closed class of verbs that add information such as
 3916 tense, aspect, person, and number.

- 3917 (8.18) **Shall** we keep chasing this murderous fish?
 3918 (8.19) What the white whale was to Ahab, **has been** hinted.
 3919 (8.20) Ahab **must** use tools.
 3920 (8.21) Meditation and water **are** wedded forever.
 3921 (8.22) Toes **are** scarce among veteran blubber men.

3922 The final example is a copula verb, which is also tagged as an auxiliary in the UD
 3923 corpus.

- 3924 • **Coordinating conjunctions** (CCONJ) express relationships between two words or
 3925 phrases, which play a parallel role:

- 3926 (8.23) Meditation **and** water are wedded forever.

- 3927 • **Subordinating conjunctions** (SCONJ) link two elements, making one syntactically
 3928 subordinate to the other:

- 3929 (8.24) There is wisdom **that** is woe.

- 3930 • **Pronouns** (PRON) are words that substitute for nouns or noun phrases.

- 3931 (8.25) Be **it what it will**, I'll go to **it** laughing.
 3932 (8.26) **I** try all things, **I** achieve **what I can**.

3933 The example includes the personal pronouns *I* and *it*, as well as the relative pronoun
 3934 *what*. Other pronouns include *myself*, *somebody*, and *nothing*.

- 3935 • **Determiners** (DET) provide additional information about the nouns or noun phrases
 3936 that they modify:

3937 (8.27) What **the** white whale was to Ahab, has been hinted.

3938 (8.28) It is not down on **any** map.

3939 (8.29) I try **all** things ...

3940 (8.30) Shall we keep chasing **this** murderous fish?

3941 Determiners include articles (*the*), possessive determiners (*their*), demonstratives
 3942 (*this murderous fish*), and quantifiers (*any map*).

- 3943 • **Numerals** (NUM) are an infinite but closed class, which includes integers, fractions,
 3944 and decimals, regardless of whether spelled out or written in numerical form.

3945 (8.31) How then can this **one** small heart beat.

3946 (8.32) I am going to put him down for the **three hundredth**.

- 3947 • **Particles** (PART) are a catch-all of function words that combine with other words or
 3948 phrases, but do not meet the conditions of the other tags. In English, this includes
 3949 the infinitival *to*, the possessive marker, and negation.

3950 (8.33) Better **to** sleep with a sober cannibal than a drunk Christian.

3951 (8.34) So man's insanity is heaven's sense

3952 (8.35) It is **not** down on any map

3953 As the second example shows, the possessive marker is not considered part of the
 3954 same token as the word that it modifies, so that *man's* is split into two tokens. (Tok-
 3955 enization is described in more detail in § 8.4.) A non-English example of a particle
 3956 is the Japanese question marker *ka*, as in,⁴

3957 (8.36) *Sensei desu ka*

 Teacher are ?

3958 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.

3959 **Other** The remaining UD tags include punctuation (PUN) and symbols (SYM). Punc-
 3960 tuation is purely structural — e.g., commas, periods, colons — while symbols can carry
 3961 content of their own. Examples of symbols include dollar and percentage symbols, math-
 3962 ematical operators, emoticons, emojis, and internet addresses. A final catch-all tag is X,
 3963 which is used for words that cannot be assigned another part-of-speech category. The X
 3964 tag is also used in cases of **code switching** (between languages), described in § 8.5.

3965 **8.1.1.2 Other tagsets**

3966 Prior to the Universal Dependency treebank, part-of-speech tagging was performed us-
 3967 ing language-specific tagsets. The dominant tagset for English was designed as part of
 3968 the **Penn Treebank** (PTB), and it includes 45 tags — more than three times as many as
 3969 the UD tagset. This granularity is reflected in distinctions between singular and plural
 3970 nouns, verb tenses and aspects, possessive and non-possessive pronouns, comparative
 3971 and superlative adjectives and adverbs (e.g., *faster, fastest*), and so on. The Brown corpus
 3972 includes a tagset that is even more detailed, with 87 tags Francis (1964), including special
 3973 tags for individual auxiliary verbs such as *be, do, and have*.

3974 Different languages make different distinctions, and so the PTB and Brown tagsets are
 3975 not appropriate for a language such as Chinese, which does not mark the verb tense (Xia,
 3976 2000); nor for Spanish, which marks every combination of person and number in the
 3977 verb ending; nor for German, which marks the case of each noun phrase. Each of these
 3978 languages requires more detail than English in some areas of the tagset, and less in other
 3979 areas. The strategy of the Universal Dependencies corpus is to design a coarse-grained
 3980 tagset to be used across all languages, and then to additionally annotate language-specific
 3981 **morphosyntactic attributes**, such as number, tense, and case. The attribute tagging task
 3982 is described in more detail in § 8.2.

3983 Social media such as Twitter have been shown to require tagsets of their own (Gimpel
 3984 et al., 2011). Such corpora contain some tokens that are not equivalent to anything en-
 3985 countered in a typical written corpus: e.g., emoticons, URLs, and hashtags. Social media
 3986 also includes dialectal words like *gonna* ('going to', e.g. *We gonna be fine*) and *Ima* ('I'm
 3987 going to', e.g., *Ima tell you one more time*), which can be analyzed either as non-standard
 3988 orthography (making tokenization impossible), or as lexical items in their own right. In
 3989 either case, it is clear that existing tags like NOUN and VERB cannot handle cases like *Ima*,
 3990 which combine aspects of the noun and verb. Gimpel et al. (2011) therefore propose a new
 3991 set of tags to deal with these cases.

3992 **8.1.2 Accurate part-of-speech tagging**

3993 Part-of-speech tagging is the problem of selecting the correct tag for each word in a sen-
 3994 tence. Success is typically measured by accuracy on an annotated test set, which is simply
 3995 the fraction of tokens that were tagged correctly.

3996 8.1.2.1 Baselines

3997 A simple baseline for part-of-speech tagging is to choose the most common tag for each
3998 word. For example, in the Universal Dependencies treebank, the word *talk* appears 96
3999 times, and 85 of those times it is labeled as a VERB: therefore, this baseline will always
4000 predict VERB for this word. For words that do not appear in the training corpus, the base-
4001 line simply guesses the most common tag overall, which is NOUN. In the Penn Treebank,
4002 this simple baseline obtains accuracy above 92%. A more rigorous evaluation is the accu-
4003 racy on **out-of-vocabulary words**, which are not seen in the training data. Tagging these
4004 words correctly requires attention to the context and the word's internal structure.

4005 8.1.2.2 Contemporary approaches

4006 Conditional random fields and structured perceptron perform at or near the state-of-the-
4007 art for part-of-speech tagging in English. For example, (Collins, 2002) achieved 97.1%
4008 accuracy on the Penn Treebank, using a structured perceptron with the following base
4009 features (originally introduced by Ratnaparkhi (1996)):

- 4010 • current word, w_m
- 4011 • previous words, w_{m-1}, w_{m-2}
- 4012 • next words, w_{m+1}, w_{m+2}
- 4013 • previous tag, y_{m-1}
- 4014 • previous two tags, (y_{m-1}, y_{m-2})
- 4015 • for rare words:
 - 4016 – first k characters, up to $k = 4$
 - 4017 – last k characters, up to $k = 4$
 - 4018 – whether w_m contains a number, uppercase character, or hyphen.

4019 Similar results for the PTB data have been achieved using conditional random fields (CRFs;
4020 Toutanova et al., 2003).

4021 More recent work has demonstrated the power of neural sequence models, such as the
4022 **long short-term memory (LSTM)** (§ 7.6). Plank et al. (2016) apply a CRF and a bidirec-
4023 tional LSTM to twenty-two languages in the UD corpus, achieving an average accuracy
4024 of 94.3% for the CRF, and 96.5% with the bi-LSTM. Their neural model employs three
4025 types of embeddings: fine-tuned word embeddings, which are updated during training;
4026 pre-trained word embeddings, which are never updated, but which help to tag out-of-
4027 vocabulary words; and character-based embeddings. The character-based embeddings
4028 are computed by running an LSTM on the individual characters in each word, thereby
4029 capturing common orthographic patterns such as prefixes, suffixes, and capitalization.
4030 Extensive evaluations show that these additional embeddings are crucial to their model's
4031 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.

4032 8.2 Morphosyntactic Attributes

4033 There is considerably more to say about a word than whether it is a noun or a verb: in En-
 4034 glish, verbs are distinguish by features such tense and aspect, nouns by number, adjectives
 4035 by degree, and so on. These features are language-specific: other languages distinguish
 4036 other features, such as **case** (the role of the noun with respect to the action of the sen-
 4037 tence, which is marked in languages such as Latin and German⁵) and **evidentiality** (the
 4038 source of information for the speaker’s statement, which is marked in languages such as
 4039 Turkish). In the UD corpora, these attributes are annotated as feature-value pairs for each
 4040 token.⁶

4041 An example is shown in Figure 8.1. The determiner *the* is marked with two attributes:
 4042 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).

4043 miner or pronominal modifier), and DEFINITE=DEF, which indicates that it is a **definite**
4044 **article** (referring to a specific, known entity). The verbs are each marked with several
4045 attributes. The auxiliary verb *was* is third-person, singular, past tense, finite (conjugated),
4046 and indicative (describing an event that has happened or is currently happenings); the
4047 main verb *destroyed* is in participle form (so there is no additional person and number
4048 information), past tense, and passive voice. Some, but not all, of these distinctions are
4049 reflected in the PTB tags VBD (past-tense verb) and VBN (past participle).

4050 While there are thousands of papers on part-of-speech tagging, there is comparatively
4051 little work on automatically labeling morphosyntactic attributes. Faruqui et al. (2016)
4052 train a support vector machine classification model, using a minimal feature set that in-
4053 cludes the word itself, its prefixes and suffixes, and type-level information listing all pos-
4054 sible morphosyntactic attributes for each word and its neighbors. Mueller et al. (2013) use
4055 a conditional random field (CRF), in which the tag space consists of all observed com-
4056 binations of morphosyntactic attributes (e.g., the tag would be DEF+ART for the word
4057 *the* in Figure 8.1). This massive tag space is managed by decomposing the feature space
4058 over individual attributes, and pruning paths through the trellis. More recent work has
4059 employed bidirectional LSTM sequence models. For example, Pinter et al. (2017) train
4060 a bidirectional LSTM sequence model. The input layer and hidden vectors in the LSTM
4061 are shared across attributes, but each attribute has its own output layer, culminating in
4062 a softmax over all attribute values, e.g. $y_t^{\text{NUMBER}} \in \{\text{SING}, \text{PLURAL}, \dots\}$. They find that
4063 character-level information is crucial, especially when the amount of labeled data is lim-
4064 ited.

4065 Evaluation is performed by first computing recall and precision for each attribute.
4066 These scores can then be averaged at either the type or token level to obtain micro- or
4067 macro-*F*-MEASURE. Pinter et al. (2017) evaluate on 23 languages in the UD treebank,
4068 reporting a median micro-*F*-MEASURE of 0.95. Performance is strongly correlated with the
4069 size of the labeled dataset for each language, with a few outliers: for example, Chinese is
4070 particularly difficult, because although the dataset is relatively large (10^5 tokens in the UD
4071 1.4 corpus), only 6% of tokens have any attributes, offering few useful labeled instances.

4072 8.3 Named Entity Recognition

4073 A classical problem in information extraction is to recognize and extract mentions of
4074 **named entities** in text. In news documents, the core entity types are people, locations, and
4075 organizations; more recently, the task has been extended to include amounts of money,
4076 percentages, dates, and times. In item 8.37 (Figure 8.2), the named entities include: *The*
4077 *U.S. Army*, an organization; *Atlanta*, a location; and *May 14, 1864*, a date. Named en-
4078 tity recognition is also a key task in **biomedical natural language processing**, with entity
4079 types including proteins, DNA, RNA, and cell lines (e.g., Collier et al., 2000; Ohta et al.,
4080 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).

4081 stracts.

4082 A standard approach to tagging named entity spans is to use discriminative sequence
 4083 labeling methods such as conditional random fields. However, the named entity recogni-
 4084 tion (NER) task would seem to be fundamentally different from sequence labeling tasks
 4085 like part-of-speech tagging: rather than tagging each token, the goal is to recover *spans*
 4086 of tokens, such as *The United States Army*.

4087 This is accomplished by the **BIO notation**, shown in Figure 8.2. Each token at the
 4088 beginning of a name span is labeled with a B- prefix; each token within a name span is la-
 4089 beled with an I- prefix. These prefixes are followed by a tag for the entity type, e.g. B-LOC
 4090 for the beginning of a location, and I-PROTEIN for the inside of a protein name. Tokens
 4091 that are not parts of name spans are labeled as O. From this representation, the entity
 4092 name spans can be recovered unambiguously. This tagging scheme is also advantageous
 4093 for learning: tokens at the beginning of name spans may have different properties than
 4094 tokens within the name, and the learner can exploit this. This insight can be taken even
 4095 further, with special labels for the last tokens of a name span, and for unique tokens in
 4096 name spans, such as *Atlanta* in the example in Figure 8.2. This is called BILOU notation,
 4097 and it can yield improvements in supervised named entity recognition (Ratinov and Roth,
 4098 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*)

4099 Another source of features is to use **gazetteers**: lists of known entity names. For example,
 4100 the U.S. Social Security Administration provides a list of tens of thousands of given names

- (1) 日文 章魚 怎麼 說?
 Japanese octopus how say
 How to say octopus in Japanese?
- (2) 日 文章 魚 怎麼 說?
 Japan essay fish how say

Figure 8.3: An example of tokenization ambiguity in Chinese (Sproat et al., 1996)

4101 — more than could be observed in any annotated corpus. Tokens or spans that match an
 4102 entry in a gazetteer can receive special features; this provides a way to incorporate hand-
 4103 crafted resources such as name lists in a learning-driven framework.

4104 **Neural sequence labeling for NER** Current research has emphasized neural sequence
 4105 labeling, using similar LSTM models to those employed in part-of-speech tagging (Ham-
 4106 merton, 2003; Huang et al., 2015; Lample et al., 2016). The bidirectional LSTM-CRF (Fig-
 4107 ure 7.4 in § 7.6) does particularly well on this task, due to its ability to model tag-to-tag
 4108 dependencies. However, Strubell et al. (2017) show that **convolutional neural networks**
 4109 can be equally accurate, with significant improvement in speed due to the efficiency of im-
 4110 plementing ConvNets on **graphics processing units (GPUs)**. The key innovation in this
 4111 work was the use of **dilated convolutions**, which are described in more detail in § 3.4.

4112 8.4 Tokenization

4113 A basic problem for text analysis, first discussed in § 4.3.1, is to break the text into a se-
 4114 quence of discrete tokens. For alphabetic languages such as English, deterministic scripts
 4115 suffice to achieve accurate tokenization. However, in logographic writing systems such
 4116 as Chinese script, words are typically composed of a small number of characters, with-
 4117 out intervening whitespace. The tokenization must be determined by the reader, with
 4118 the potential for occasional ambiguity, as shown in Figure 8.3. One approach is to match
 4119 character sequences against a known dictionary (e.g., Sproat et al., 1996), using additional
 4120 statistical information about word frequency. However, no dictionary is completely com-
 4121 prehensive, and dictionary-based approaches can struggle with such out-of-vocabulary
 4122 words.

4123 Chinese tokenization has therefore been approached as a supervised sequence label-
 4124 ing problem. Xue et al. (2003) train a logistic regression classifier to make independent
 4125 segmentation decisions while moving a sliding window across the document. A set of
 4126 rules is then used to convert these individual classification decisions into an overall tok-
 4127 enization of the input. However, these individual decisions may be globally suboptimal,
 4128 motivating a structure prediction approach. Peng et al. (2004) train a conditional random

4129 field to predict labels of START or NONSTART on each character. More recent work has
 4130 employed neural network architectures. For example, Chen et al. (2015) use an LSTM-
 4131 CRF architecture, as described in § 7.6: they construct a trellis, in which each tag is scored
 4132 according to the hidden state of an LSTM, and tag-tag transitions are scored according
 4133 to learned transition weights. The best-scoring segmentation is then computed by the
 4134 Viterbi algorithm.

4135 8.5 Code switching

4136 Multilingual speakers and writers do not restrict themselves to a single language. **Code**
4137 **switching** is the phenomenon of switching between languages in speech and text (Auer,
4138 2013; Poplack, 1980). Written code switching has become more common in online social
4139 media, as in the following extract from Justin Trudeau's website:⁷

- 4140 (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*

4142 Accurately analyzing such texts requires first determining which languages are being
4143 used. Furthermore, quantitative analysis of code switching can provide insights on the
4144 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)

4160 8.6 Dialogue acts

4161 The sequence labeling problems that we have discussed so far have been over sequences
 4162 of word tokens or characters (in the case of tokenization). However, sequence labeling
 4163 can also be performed over higher-level units, such as **utterances**. **Dialogue acts** are la-
 4164 bels over utterances in a dialogue, corresponding roughly to the speaker’s intention —
 4165 the utterance’s **illocutionary force** (Austin, 1962). For example, an utterance may state a
 4166 proposition (*it is not down on any map*), pose a question (*shall we keep chasing this murderous*
 4167 *fish?*), or provide a response (*aye aye!*). Stolcke et al. (2000) describe how a set of 42 dia-
 4168 logue acts were annotated for the 1,155 conversations in the Switchboard corpus (Godfrey
 4169 et al., 1992).⁸

4170 An example is shown in Figure 8.4. The annotation is performed over UTTERANCES,
 4171 with the possibility of multiple utterances per **conversational turn** (in cases such as inter-
 4172 ruptions, an utterance may split over multiple turns). Some utterances are clauses (e.g., *So*
 4173 *do you go to college right now?*), while others are single words (e.g., *yeah*). Stolcke et al. (2000)
 4174 report that hidden Markov models (HMMs) achieve 96% accuracy on supervised utter-
 4175 ance segmentation. The labels themselves reflect the conversational goals of the speaker:
 4176 the utterance *yeah* functions as an answer in response to the question *you’re a senior now*,
 4177 but in the final line of the excerpt, it is a **backchannel** (demonstrating comprehension).

4178 For task of dialogue act labeling, Stolcke et al. (2000) apply a hidden Markov model.
 4179 The probability $p(w_m | y_m)$ must generate the entire sequence of words in the utterance,
 4180 and it is modeled as a trigram language model (§ 6.1). Stolcke et al. (2000) also account
 4181 for acoustic features, which capture the **prosody** of each utterance — for example, tonal
 4182 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.

4183 as questions and answers. These features are handled with an additional emission distri-
4184 bution, $p(a_m | y_m)$, which is modeled with a probabilistic decision tree (Murphy, 2012).
4185 While acoustic features yield small improvements overall, they play an important role in
4186 distinguish questions from statements, and agreements from backchannels.

4187 Recurrent neural architectures for dialogue act labeling have been proposed by Kalch-
4188 brenner and Blunsom (2013) and Ji et al. (2016), with strong empirical results. Both models
4189 are recurrent at the utterance level, so that each complete utterance updates a hidden state.
4190 The recurrent-convolutional network of Kalchbrenner and Blunsom (2013) uses convolu-
4191 tion to obtain a representation of each individual utterance, while Ji et al. (2016) use a
4192 second level of recurrence, over individual words. This enables their method to also func-
4193 tion as a language model, giving probabilities over sequences of words in a document.

4194 **Exercises**

- 4195 1. [todo: exercises tk]

4196 **Chapter 9**

4197 **Formal language theory**

4198 We have now seen methods for learning to label individual words, vectors of word counts,
4199 and sequences of words; we will soon proceed to more complex structural transfor-
4200 mations. Most of these techniques could apply to counts or sequences from any discrete vo-
4201 cabulary; there is nothing fundamentally linguistic about, say, a hidden Markov model.
4202 This raises a basic question that this text has not yet considered: what is a language?

4203 This chapter will take the perspective of **formal language theory**, in which a language
4204 is defined as a set of **strings**, each of which is a sequence of elements from a finite alphabet.
4205 For interesting languages, there are an infinite number of strings that are in the language,
4206 and an infinite number of strings that are not. For example:

- 4207 • the set of all even-length sequences from the alphabet $\{a, b\}$, e.g., $\{\emptyset, aa, ab, ba, bb, aaaa, aaab, \dots\}$;
- 4208 • the set of all sequences from the alphabet $\{a, b\}$ that contain *aaa* as a substring, e.g.,
4209 $\{aaa, aaaa, baaa, aaab, \dots\}$;
- 4210 • the set of all sequences of English words (drawn from a finite dictionary) that con-
4211 tain at least one verb (a finite subset of the dictionary);
- 4212 • the `python` programming language.

4213 Formal language theory defines classes of languages and their computational prop-
4214 erties. Of particular interest is the computational complexity of solving the **membership**
4215 **problem** — determining whether a string is in a language. The chapter will focus on
4216 three classes of formal languages: regular, context-free, and “mildly” context-sensitive
4217 languages.

4218 A key insight of 20th century linguistics is that formal language theory can be usefully
4219 applied to natural languages such as English, by designing formal languages that cap-
4220 ture as many properties of the natural language as possible. For many such formalisms, a
4221 useful linguistic analysis comes as a byproduct of solving the membership problem. The

membership problem can be generalized to the problems of *scoring* strings for their acceptability (as in language modeling), and of **transducing** one string into another (as in translation).

9.1 Regular languages

Sooner or later, most computer scientists will write a **regular expression**. If you have, then you have defined a **regular language**, which is any language that can be defined by a regular expression. Formally, a regular expression can include the following elements:

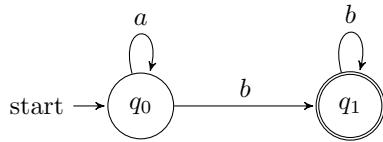
- A **literal character** drawn from some finite alphabet Σ .
- The **empty string** ϵ .
- The concatenation of two regular expressions RS , where R and S are both regular expressions. The resulting expression accepts any string that can be decomposed $x = yz$, where y is accepted by R and z is accepted by S .
- The alternation $R \mid S$, where R and S are both regular expressions. The resulting expression accepts a string x if it is accepted by R or it is accepted by S .
- The **Kleene star** R^* , which accepts any string x that can be decomposed into a sequence of strings which are all accepted by R .
- Parenthesization ((R)), which is used to limit the scope of the concatenation, alternation, and Kleene star operators.

Here are some example regular expressions:

- The set of all even length strings on the alphabet $\{a, b\}$: $((aa)|(ab)|(ba)|(bb))^*$
- The set of all sequences of the alphabet $\{a, b\}$ that contain aaa as a substring: $(a|b)^*aaa(a|b)^*$
- The set of all sequences of English words that contain at least one verb: W^*VW^* , where W is an alternation between all words in the dictionary, and V is an alternation between all verbs ($V \subseteq W$).

This list does not include a regular expression for the Python programming language, because this language is not regular — there is no regular expression that can capture its syntax. We will discuss why towards the end of this section.

Regular languages are **closed** under union, intersection, and concatenation. This means, for example, that if two languages L_1 and L_2 are regular, then so are the languages $L_1 \cup L_2$, $L_1 \cap L_2$, and the language of strings that can be decomposed as $s = tu$, with $s \in L_1$ and $t \in L_2$. Regular languages are also closed under negation: if L is regular, then so is the language $\bar{L} = \{s \notin L\}$.

Figure 9.1: State diagram for the finite state acceptor M_1 .

4254 **9.1.1 Finite state acceptors**

4255 A regular expression defines a regular language, but does not give an algorithm for de-
 4256 termining whether a string is in the language that it defines. **Finite state automata** are
 4257 theoretical models of computation on regular languages, which involve transitions be-
 4258 tween a finite number of states. The most basic type of finite state automaton is the **finite**
 4259 **state acceptor (FSA)**, which describes the computation involved in testing if a string is
 4260 a member of a language. Formally, a finite state acceptor is a tuple $M = (Q, \Sigma, q_0, F, \delta)$,
 4261 consisting of:

- 4262 • a finite alphabet Σ of input symbols;
- 4263 • a finite set of states $Q = \{q_0, q_1, \dots, q_n\}$;
- 4264 • a start state $q_0 \in Q$;
- 4265 • a set of final states $F \subseteq Q$;
- 4266 • a transition function $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$. The transition function maps from a
 4267 state and an input symbol (or empty string ϵ) to a *set* of possible resulting states.

4268 A **path** in M is a sequence of transitions, $\pi = t_1, t_2, \dots, t_N$, where each t_i traverses an
 4269 arc in the transition function δ . The finite state acceptor M accepts a string ω if there is
 4270 a **accepting path**, in which the initial transition t_1 begins at the start state q_0 , the final
 4271 transition t_N terminates in a final state in Q , and the entire input ω is consumed.

4272 **9.1.1.1 Example**

Consider the following FSA, M_1 .

$$\Sigma = \{a, b\} \quad [9.1]$$

$$Q = \{q_0, q_1\} \quad [9.2]$$

$$F = \{q_1\} \quad [9.3]$$

$$\delta = \{(q_0, a) \rightarrow q_0, (q_0, b) \rightarrow q_1, (q_1, b) \rightarrow q_1\}. \quad [9.4]$$

4273 This FSA defines a language over an alphabet of two symbols, a and b . The transition
 4274 function δ is written as a set of arcs: $(q_0, a) \rightarrow q_0$ says that if the machine is in state

4275 q_0 and reads symbol a , it stays in q_0 . Figure 9.1 provides a graphical representation of
 4276 M_1 . Because each pair of initial state and symbol has at most one resulting state, M_1 is
 4277 **deterministic**: each string ω induces at most one accepting path. Note that there are no
 4278 transitions for the symbol a in state q_1 ; if a is encountered in q_1 , then the acceptor is stuck,
 4279 and the input string is rejected.

4280 What strings does M_1 accept? The start state is q_0 , and we have to get to q_1 , since this
 4281 is the only final state. Any number of a symbols can be consumed in q_0 , but a b symbol is
 4282 required to transition to q_1 . Once there, any number of b symbols can be consumed, but
 4283 an a symbol cannot. So the regular expression corresponding to the language defined by
 4284 M_1 is a^*bb^* .

4285 9.1.1.2 Computational properties of finite state acceptors

4286 The key computational question for finite state acceptors is: how fast can we determine
 4287 whether a string is accepted? For deterministic FSAs, this computation can be performed
 4288 by Dijkstra's algorithm, with time complexity $\mathcal{O}(V \log V + E)$, where V is the number of
 4289 vertices in the FSA, and E is the number of edges (Cormen et al., 2009). Non-deterministic
 4290 FSAs (NFSAs) can include multiple transitions from a given symbol and state. Any NSFA
 4291 can be converted into a deterministic FSA, but the resulting automaton may have a num-
 4292 ber of states that is exponential in the number of size of the original NFSFA (Mohri et al.,
 4293 2002).

4294 9.1.2 Morphology as a regular language

4295 Many words have internal structure, such as prefixes and suffixes that shape their mean-
 4296 ing. The study of word-internal structure is the domain of **morphology**, of which there
 4297 are two main types:

- 4298 • **Derivational morphology** describes the use of affixes to convert a word from one
 4299 grammatical category to another (e.g., from the noun *grace* to the adjective *graceful*),
 4300 or to change the meaning of the word (e.g., from *grace* to *disgrace*).
- 4301 • **Inflectional morphology** describes the addition of details such as gender, number,
 4302 person, and tense (e.g., the *-ed* suffix for past tense in English).

4303 Morphology is a rich topic in linguistics, deserving of a course in its own right.¹ The
 4304 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).

4305 current section deals with derivational morphology; inflectional morphology is discussed
 4306 in § 9.1.4.3.

4307 Suppose that we want to write a program that accepts only those words that are con-
 4308 structed in accordance with the rules of English derivational morphology:

- 4309 (9.1) grace, graceful, gracefully, *gracelyful
- 4310 (9.2) disgrace, *ungrace, disgraceful, disgracefully
- 4311 (9.3) allure, *allureful, alluring, alluringly
- 4312 (9.4) fairness, unfair, *disfair, fairly

4313 (Recall that the asterisk indicates that a linguistic example is judged unacceptable by flu-
 4314 ent speakers of a language.) These examples cover only a tiny corner of English deriva-
 4315 tional morphology, but a number of things stand out. The suffix *-ful* converts the nouns
 4316 *grace* and *disgrace* into adjectives, and the suffix *-ly* converts adjectives into adverbs. These
 4317 suffixes must be applied in the correct order, as shown by the unacceptability of **grace-
 4318 lyful*. The *-ful* suffix works for only some words, as shown by the use of *alluring* as the
 4319 adjectival form of *allure*. Other changes are made with prefixes, such as the derivation
 4320 of *disgrace* from *grace*, which roughly corresponds to a negation; however, *fair* is negated
 4321 with the *un-* prefix instead. Finally, while the first three examples suggest that the direc-
 4322 tion of derivation is noun → adjective → adverb, the example of *fair* suggests that the
 4323 adjective can also be the base form, with the *-ness* suffix performing the conversion to a
 4324 noun.

4325 Can we build a computer program that accepts only well-formed English words, and
 4326 rejects all others? This might at first seem trivial to solve with a brute-force attack: simply
 4327 make a dictionary of all valid English words. But such an approach fails to account for
 4328 morphological **productivity** — the applicability of existing morphological rules to new
 4329 words and names, such as *Trump* to *Trumpy* and *Trumpkin*, and *Clinton* to *Clintonian* and
 4330 *Clintonite*. We need an approach that represents morphological rules explicitly, and for
 4331 this we will try a finite state acceptor.

4332 The dictionary approach can be implemented as a finite state acceptor, with the vo-
 4333 cabulary Σ equal to the vocabulary of English, and a transition from the start state to the
 4334 accepting state for each word. But this would of course fail to generalize beyond the origi-
 4335 nal vocabulary, and would not capture anything about the **morphotactic** rules that govern
 4336 derivations from new words. The first step towards a more general approach is shown in
 4337 Figure 9.2, which is the state diagram for a finite state acceptor in which the vocabulary
 4338 consists of **morphemes**, which include **stems** (e.g., *grace*, *allure*) and **affixes** (e.g., *dis-*, *-ing*,
 4339 *-ly*). This finite state acceptor consists of a set of paths leading away from the start state,
 4340 with derivational affixes added along the path. Except for q_{neg} , the states on these paths
 4341 are all final, so the FSA will accept *disgrace*, *disgraceful*, and *disgracefully*, but not *dis-*.

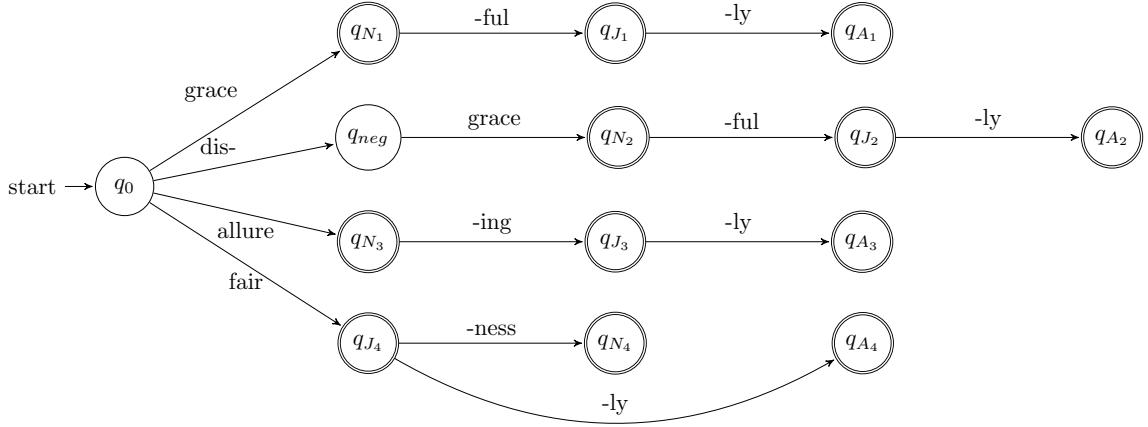


Figure 9.2: A finite state acceptor for a fragment of English derivational morphology. Each path represents possible derivations from a single root form.

4342 This FSA can be **minimized** to the form shown in Figure 9.3, which makes the general-
 4343 ity of the finite state approach more apparent. For example, the transition from q_0 to
 4344 q_{J_2} can be made to accept not only *fair* but any single-morpheme (**monomorphemic**) ad-
 4345 jective that takes *-ness* and *-ly* as suffixes. In this way, the finite state acceptor can easily
 4346 be extended: as new word stems are added to the vocabulary, their derived forms will be
 4347 accepted automatically. Of course, this FSA would still need to be extended considerably
 4348 to cover even this small fragment of English morphology. As shown by cases like *music*
 4349 → *musical*, *athlete* → *athletic*, English includes several classes of nouns, each with its own
 4350 rules for derivation.

4351 The FSAs shown in Figure 9.2 and 9.3 accept *allureing*, not *alluring*. This reflects a dis-
 4352 tinction between morphology — the question of which morphemes to use, and in what
 4353 order — and **orthography** — the question of how the morphemes are rendered in written
 4354 language. Just as orthography requires dropping the *e* preceding the *-ing* suffix, **phonol-**
 4355 **ogy** imposes a related set of constraints on how words are rendered in speech. As we will
 4356 see soon, these issues are handled through **finite state transducers**, which are finite state
 4357 automata that take inputs and produce outputs.

4358 9.1.3 Weighted finite state acceptors

4359 According to the FSA treatment of morphology, every word is either in or out of the lan-
 4360 guage, with no wiggle room. Perhaps you agree that *musicky* and *fishful* are not valid
 4361 English words; but if forced to choose, you probably find *a fishful stew* or *a musicky trib-*
 4362 *ute* preferable to *behaving disgracelyful*. Rather than asking whether a word is acceptable,
 4363 we might like to ask how acceptable it is. Aronoff (1976, page 36) puts it another way:

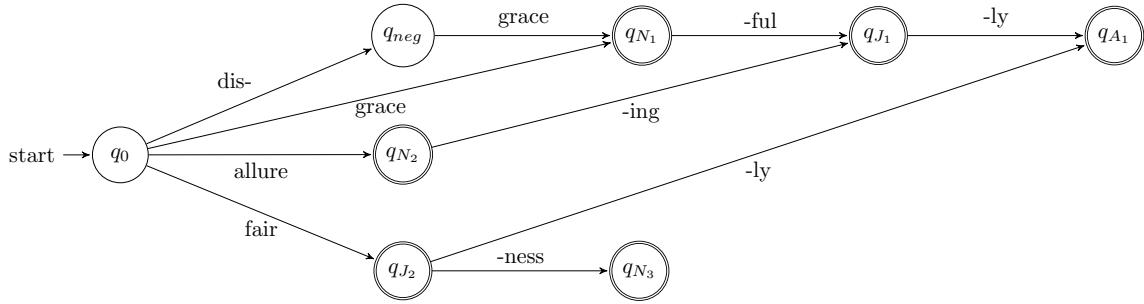


Figure 9.3: Minimization of the finite state acceptor shown in Figure 9.2.

4364 “Though many things are possible in morphology, some are more possible than others.”
 4365 But finite state acceptors give no way to express preferences among technically valid
 4366 choices.

4367 **Weighted finite state acceptors (WFSAs)** are generalizations of FSAs, in which each
 4368 accepting path is assigned a score, computed from the transitions, the initial state, and the
 4369 final state. Formally, a weighted finite state acceptor $M = (Q, \Sigma, \lambda, \rho, \delta)$ consists of:

- 4370 • a finite set of states $Q = \{q_0, q_1, \dots, q_n\}$;
- 4371 • a finite alphabet Σ of input symbols;
- 4372 • an initial weight function, $\lambda : Q \mapsto \mathbb{R}$;
- 4373 • a final weight function $\rho : Q \mapsto \mathbb{R}$;
- 4374 • a transition function $\delta : Q \times \Sigma \times Q \mapsto \mathbb{R}$.

4375 WFSAs depart from the FSA formalism in three ways: every state can be an initial
 4376 state, with score $\lambda(q)$; every state can be an accepting state, with score $\rho(q)$; transitions are
 4377 possible between any pair of states on any input, with a score $\delta(q_i, \omega, q_j)$. Nonetheless,
 4378 FSAs can be viewed as a special case: for any FSA M we can build an equivalent WFSA
 4379 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
 4380 transitions $\{(q_1, \omega) \rightarrow q_2\}$ that are not permitted by the transition function of M .

4381 The total score for any path $\pi = t_1, t_2, \dots, t_N$ is equal to the sum of these scores,

$$d(\pi) = \lambda(\text{from-state}(t_1)) + \sum_n^N \delta(t_n) + \rho(\text{to-state}(t_N)). \quad [9.5]$$

4382 A **shortest-path algorithm** is used to find the minimum-cost path through a WFSA for
 4383 string ω , with time complexity $\mathcal{O}(E + V \log V)$, where E is the number of edges and V is
 4384 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

4385 **9.1.3.1 N-gram language models as WFSAs**

4386 In **n-gram language models** (see § 6.1), the probability of a sequence of tokens w_1, w_2, \dots, w_M
 4387 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:

$$\begin{aligned} \delta(q_i, \omega, q_j) &= \begin{cases} \log \Pr(w_m = j | w_{m-1} = i), & \omega = j \\ -\infty, & \omega \neq j \end{cases} \\ \lambda(q_i) &= \log \Pr(w_1 = i | w_0 = \square) \\ \rho(q_i) &= \log \Pr(w_{M+1} = \blacksquare | w_M = i). \end{aligned}$$

4388 The transition function is designed to ensure that the context is recorded accurately:
 4389 we can move to state j on input ω only if $\omega = j$; otherwise, transitioning to state j is
 4390 forbidden by the weight of $-\infty$. The initial weight function $\lambda(q_i)$ is the log probability of
 4391 receiving i as the first token, and the final weight function $\rho(q_i)$ is the log probability of
 4392 receiving an “end-of-string” token after observing $w_M = i$.

4393 **9.1.3.2 *Semiring weighted finite state acceptors**

4394 The n -gram language model WFSA is deterministic: each input has exactly one accepting
 4395 path, for which the WFSA computes a score. In non-deterministic WFSAs, a given input

probabilities, so we want the path with the maximum score, which can be accomplished by making each local score into a **negative** log-probability. The remainder of this section will refer to **best-path algorithms**, which are assumed to “do the right thing.”

4396 may have multiple accepting paths. In some applications, the score for the input is ag-
 4397 gregated across all such paths. Such aggregate scores can be computed by generalizing
 4398 WFSAs with **semiring notation**, first introduced in § 7.7.3.

4399 Let $d(\pi)$ represent the total score for path $\pi = t_1, t_2, \dots, t_N$, which is computed as,

$$d(\pi) = \lambda(\text{from-state}(t_1)) \otimes \delta(t_1) \otimes \delta(t_2) \otimes \dots \otimes \delta(t_N) \otimes \rho(\text{to-state}(t_N)). \quad [9.8]$$

4400 This is a generalization of Equation 9.5 to semiring notation, using the semiring multipli-
 4401 cation operator \otimes in place of addition.

4402 Now let $s(\omega)$ represent the total score for all paths $\Pi(\omega)$ that consume input ω ,

$$s(\omega) = \bigoplus_{\pi \in \Pi(\omega)} d(\pi). \quad [9.9]$$

4403 Here, semiring addition (\oplus) is used to combine the scores of multiple paths.

4404 The generalization to semirings covers a number of useful special cases. In the log-
 4405 probability semiring, multiplication is defined as $\log p(x) \otimes \log p(y) = \log p(x) + \log p(y)$,
 4406 and addition is defined as $\log p(x) \oplus \log p(y) = \log(p(x) + p(y))$. Thus, $s(\omega)$ represents
 4407 the log-probability of accepting input ω , marginalizing over all paths $\pi \in \Pi(\omega)$. In the
 4408 **boolean semiring**, the \otimes operator is logical conjunction, and the \oplus operator is logical
 4409 disjunction. This reduces to the special case of unweighted finite state acceptors, where
 4410 the score $s(\omega)$ is a boolean indicating whether there exists any accepting path for ω . In
 4411 the **tropical semiring**, the \oplus operator is a maximum, so the resulting score is the score of
 4412 the best-scoring path through the WFSAs. The OpenFST toolkit uses semirings and poly-
 4413 morphism to implement general algorithms for weighted finite state automata (Allauzen
 4414 et al., 2007).

4415 9.1.3.3 *Interpolated n -gram language models

4416 Recall from § 6.2.3 that an **interpolated n -gram language model** combines the probabili-
 4417 ties from multiple n -gram models. For example, an interpolated bigram language model
 4418 computes probability,

$$\hat{p}(w_m | w_{m-1}) = \lambda_1 p_1(w_m) + \lambda_2 p_2(w_m | w_{m-1}), \quad [9.10]$$

4419 with \hat{p} indicating the interpolated probability, p_2 indicating the bigram probability, and
 4420 p_1 indicating the unigram probability. We set $\lambda_2 = (1 - \lambda_1)$ so that the probabilities sum
 4421 to one.

4422 Interpolated bigram language models can be implemented using a non-deterministic
 4423 WFSAs (Knight and May, 2009). The basic idea is shown in Figure 9.4. In an interpolated
 4424 bigram language model, there is one state for each element in the vocabulary — in this

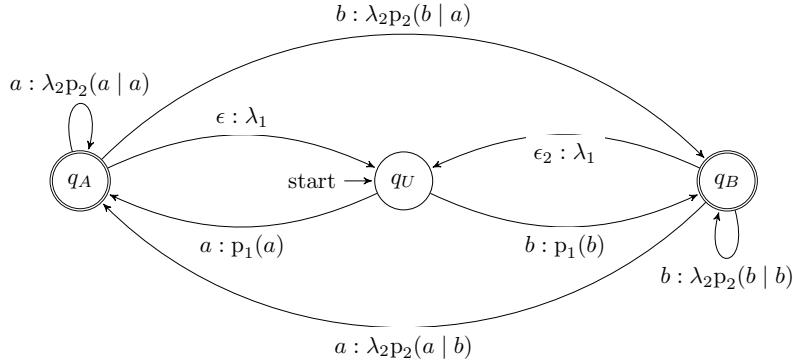


Figure 9.4: WFSA implementing an interpolated bigram/unigram language model, on the alphabet $\Sigma = \{a, b\}$. For simplicity, the WFSA is constrained to force the first token to be generated from the unigram model, and does not model the emission of the end-of-sequence token.

4425 case, the states q_A and q_B — which capture the contextual conditioning in the bigram
 4426 probabilities. To model unigram probabilities, there is an additional state q_U , which “for-
 4427 gets” the context. Transitions out of q_U involve unigram probabilities, $p_1(a)$ and $p_2(b)$;
 4428 transitions into q_U emit the empty symbol ϵ , and have probability λ_1 , reflecting the inter-
 4429 polation weight for the unigram model. The interpolation weight for the bigram model is
 4430 included in the weight of the transition $q_A \rightarrow q_B$.

4431 The epsilon transitions into q_U make this WFSA non-deterministic. Consider the score
 4432 for the sequence (a, b, b) . The initial state is q_U , so the symbol a is generated with score
 4433 $p_1(a)$ ³ Next, we can generate b from the unigram model by taking the transition $q_A \rightarrow q_B$,
 4434 with score $\lambda_2 p_2(b | a)$. Alternatively, we can take a transition back to q_U with score λ_1 ,
 4435 and then emit b from the unigram model with score $p_1(b)$. To generate the final b token,
 4436 we face the same choice: emit it directly from the self-transition to q_B , or transition to q_U
 4437 first.

The total score for the sequence (a, b, b) is the semiring sum over all accepting paths,

$$\begin{aligned}
 s(a, b, b) &= (p_1(a) \otimes \lambda_2 p_2(b | a) \otimes \lambda_2 p_2(b | b)) \\
 &\oplus (p_1(a) \otimes \lambda_1 \otimes p_1(b) \otimes \lambda_2 p_2(b | b)) \\
 &\oplus (p_1(a) \otimes \lambda_2 p_2(b | a) \otimes p_1(b) \otimes p_1(b)) \\
 &\oplus (p_1(a) \otimes \lambda_1 \otimes p_1(b) \otimes p_1(b) \otimes p_1(b)). \tag{[9.11]}
 \end{aligned}$$

4438 Each line in Equation 9.11 represents the probability of a specific path through the WFSA.
 4439 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.

4440 transition weight, which are themselves probabilities. The \oplus operator is addition, so that
 4441 the total score is the sum of the scores (probabilities) for each path. This corresponds to
 4442 the probability under the interpolated bigram language model.

4443 **9.1.4 Finite state transducers**

4444 Finite state acceptors can determine whether a string is in a regular language, and weighted
 4445 finite state acceptors can compute a score for every string over a given alphabet. **Finite**
 4446 **state transducers** (FSTs) extend the formalism further, by adding an output symbol to each
 4447 transition. Formally, a finite state transducer is a tuple $T = (Q, \Sigma, \Omega, \lambda, \rho, \delta)$, with Ω repre-
 4448 senting an output vocabulary and the transition function $\delta : Q \times (\Sigma \cup \epsilon) \times (\Omega \cup \epsilon) \times Q \rightarrow \mathbb{R}$
 4449 mapping from states, input symbols, and output symbols to states. The remaining ele-
 4450 ments (Q, Σ, λ, ρ) are identical to their definition in weighted finite state acceptors (§ 9.1.3).
 4451 Thus, each path through the FST T transduces the input string into an output.

4452 **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,

$$\delta(q, a, a, q) = \delta(q, b, b, q) = 0 \quad [9.12]$$

$$\delta(q, a, b, q) = \delta(q, b, a, q) = 1 \quad [9.13]$$

$$\delta(q, a, \epsilon, q) = \delta(q, b, \epsilon, q) = 1 \quad [9.14]$$

$$\delta(q, \epsilon, a, q) = \delta(q, \epsilon, b, q) = 1. \quad [9.15]$$

4453 The state diagram is shown in Figure 9.5.

4454 For a given string pair, there are multiple paths through the transducer: the best-
 4455 scoring path from *dessert* to *desert* involves a single deletion, for a total score of 1; the
 4456 worst-scoring path involves seven deletions and six additions, for a score of 13.

4457 **9.1.4.2 The Porter stemmer**

The Porter (1980) stemming algorithm is a “lexicon-free” algorithm for stripping suffixes from English words, using a sequence of character-level rules. Each rule can be described

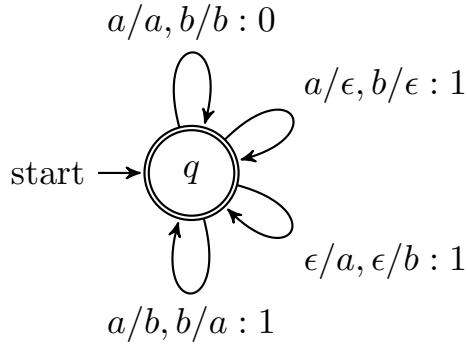


Figure 9.5: State diagram for the Levenshtein edit distance finite state transducer. The label $x/y : c$ indicates a cost of c for a transition with input x and output y .

by an unweighted finite state transducer. The first rule is:

-sses → -ss e.g., *dresses* → *dress* [9.16]

-ies → -i e.g., *parties* → *parti* [9.17]

-ss → -ss e.g., *dress* → *dress* [9.18]

-s → ε e.g., *cats* → *cat* [9.19]

4458 The final two lines appear to conflict; they are meant to be interpreted as an instruction
 4459 to remove a terminal *-s* unless it is part of an *-ss* ending. A state diagram to handle just
 4460 these final two lines is shown in Figure 9.6. Make sure you understand how this finite
 4461 state transducer handles *cats*, *steps*, *bass*, and *basses*.

4462 9.1.4.3 Inflectional morphology

4463 In **inflectional morphology**, word **lemmas** are modified to add grammatical information
 4464 such as tense, number, and case. For example, many English nouns are pluralized by the
 4465 suffix *-s*, and many verbs are converted to past tense by the suffix *-ed*. English's inflectional
 4466 morphology is considerably simpler than many of the world's languages. For example,
 4467 Romance languages (derived from Latin) feature complex systems of verb suffixes which
 4468 must agree with the person and number of the verb, as shown in Table 9.1.

4469 The task of **morphological analysis** is to read a form like *canto*, and output an analysis
 4470 like CANTAR+VERB+PRESIND+1P+SING, where +PRESIND describes the tense as present
 4471 indicative, +1P indicates the first-person, and +SING indicates the singular number. The
 4472 task of **morphological generation** is the reverse, going from CANTAR+VERB+PRESIND+1P+SING
 4473 to *canto*. Finite state transducers are an attractive solution, because they can solve both
 4474 problems with a single model (Beesley and Karttunen, 2003). As an example, Figure 9.7
 4475 shows a fragment of a finite state transducer for Spanish inflectional morphology. The

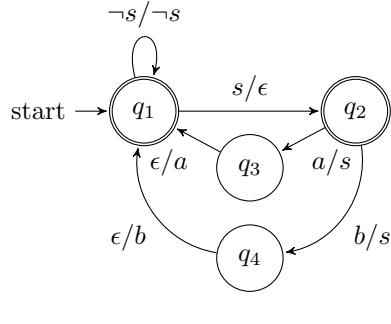


Figure 9.6: State diagram for final two lines of step 1a of the Porter stemming diagram. States q_3 and q_4 “remember” the observations a and b respectively; the ellipsis \dots represents additional states for each symbol in the input alphabet. The notation $\neg s / \neg s$ is not part of the FST formalism; 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.

4476 input vocabulary Σ corresponds to the set of letters used in Spanish spelling, and the out-
 4477 put vocabulary Ω corresponds to these same letters, plus the vocabulary of morphological
 4478 features (e.g., +SING, +VERB). In Figure 9.7, there are two paths that take *canto* as input,
 4479 corresponding to the verb and noun meanings; the choice between these paths could be
 4480 guided by a part-of-speech tagger. By **inversion**, the inputs and outputs for each trans-
 4481 transition are switched, resulting in a finite state generator, capable of producing the correct
 4482 **surface form** for any morphological analysis.

4483 Finite state morphological analyzers and other unweighted transducers can be de-
 4484 signed by hand. The designer’s goal is to avoid **overgeneration** — accepting strings or
 4485 making transductions that are not valid in the language — as well as **undergeneration** —

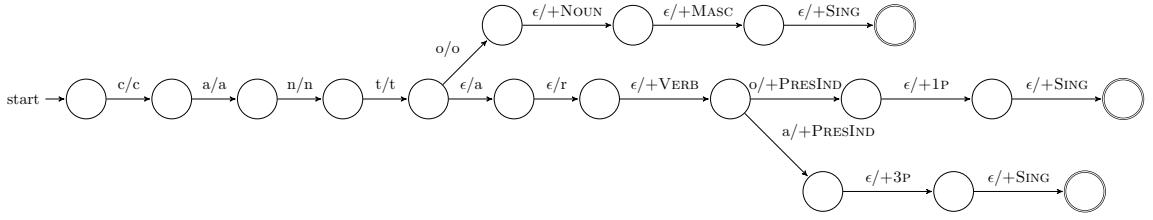


Figure 9.7: Fragment of a finite state transducer for Spanish morphology. There are two accepting paths for the input *canto*: *canto+NOUN+MASC+SING* (masculine singular noun, meaning a song), and *cantar+VERB+PRESIND+1P+SING* (I sing). There is also an accepting path for *canta*, with output *cantar+VERB+PRESIND+3P+SING* (he/she sings).

4486 failing to accept strings or transductions that are valid. For example, a pluralization trans-
 4487 ducer that does not accept *foot/feet* would undergenerate. Suppose we “fix” the transducer
 4488 to accept this example, but as a side effect, it now accepts *boot/beet*; the transducer would
 4489 then be said to overgenerate. A transducer that accepts *foot/foots* but not *foot/feet* would
 4490 both overgenerate and undergenerate.

4491 9.1.4.4 Finite state composition

4492 Designing finite state transducers to capture the full range of morphological phenomena
 4493 in any real language is a huge task. Modularization is a classic computer science approach
 4494 for this situation: decompose a large and unwieldy problem into a set of subproblems,
 4495 each of which will hopefully have a concise solution. Finite state automata can be mod-
 4496 ularized through **composition**: feeding the output of one transducer T_1 as the input to
 4497 another transducer T_2 , written $T_2 \circ T_1$. Formally, if there exists some y such that $(x, y) \in T_1$
 4498 (meaning that T_1 produces output y on input x), and $(y, z) \in T_2$, then $(x, z) \in (T_2 \circ T_1)$.
 4499 Because finite state transducers are **closed** under composition, there is guaranteed to be
 4500 a single finite state transducer that $T_3 = T_2 \circ T_1$, which can be constructed as a machine
 4501 with one state for each pair of states in T_1 and T_2 (Mohri et al., 2002).

4502 **Example: Morphology and orthography** In English morphology, the suffix *-ed* is added
 4503 to signal the past tense for many verbs: *cook*→*cooked*, *want*→*wanted*, etc. However, English
 4504 **orthography** dictates that this process cannot produce a spelling with consecutive *e*'s, so
 4505 that *bake*→*baked*, not *bakeed*. A modular solution is to build separate transducers for mor-
 4506 phology and orthography. The morphological transducer T_M transduces from *bake+PAST*
 4507 to *bake+ed*, with the *+* symbol indicating a segment boundary. The input alphabet of T_M
 4508 includes the lexicon of words and the set of morphological features; the output alphabet
 4509 includes the characters *a-z* and the *+* boundary marker. Next, an orthographic transducer
 4510 T_O is responsible for the transductions *cook+ed*→*cooked*, and *bake+ed*→*baked*. The input
 4511 alphabet of T_O must be the same as the output alphabet for T_M , and the output alphabet

4512 is simply the characters *a-z*. The composed transducer ($T_O \circ T_M$) then transduces from
 4513 *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, $p(\mathbf{w}, \mathbf{y})$, which can be computed as a path through a **trellis** structure. This trellis is itself a weighted finite state acceptor, with edges between all adjacent nodes $q_{m-1,i} \rightarrow q_{m,j}$ on input $Y_m = j$. The edge weights are log-probabilities,

$$\delta(q_{m-1,i}, Y_m = j, q_{m,j}) = \log p(w_m, Y_m = j | Y_{m-1} = i) \quad [9.20]$$

$$= \log p(w_m | Y_m = j) + \log \Pr(Y_m = j | Y_{m-1} = i). \quad [9.21]$$

4514 Because there is only one possible transition for each tag Y_m , this WFSA is deterministic.
 4515 The score for any tag sequence $\{y_m\}_{m=1}^M$ is the sum of these log-probabilities, correspond-
 4516 ing to the total log probability $\log p(\mathbf{w}, \mathbf{y})$. Furthermore, the trellis can be constructed by
 4517 the composition of simpler FSTs.

- 4518 • First, construct a “transition” transducer to represent a bigram probability model
 4519 over tag sequences, T_T . This transducer is almost identical to the n -gram language
 4520 model acceptor in § 9.1.3.1: there is one state for each tag, and the edge weights
 4521 equal to the transition log-probabilities, $\delta(q_i, j, j, q_j) = \log \Pr(Y_m = j | Y_{m-1} = i)$.
 4522 Note that T_T is a transducer, with identical input and output at each arc; this makes
 4523 it possible to compose T_T with other transducers.
- 4524 • Next, construct an “emission” transducer to represent the probability of words given
 4525 tags, T_E . This transducer has only a single state, with arcs for each word/tag pair,
 4526 $\delta(q_0, i, j, q_0) = \log \Pr(W_m = j | Y_m = i)$. The input vocabulary is the set of all tags,
 4527 and the output vocabulary is the set of all words.
- 4528 • The composition $T_E \circ T_T$ is a finite state transducer with one state per tag, as shown
 4529 in Figure 9.8. Each state has $V \times K$ outgoing edges, representing transitions to each
 4530 of the K other states, with outputs for each of the V words in the vocabulary. The
 4531 weights for these edges are equal to,

$$\delta(q_i, Y_m = j, w_m, q_j) = \log p(w_m, Y_m = j | Y_{m-1} = i). \quad [9.22]$$

- 4532 • The trellis is a structure with $M \times K$ nodes, for each of the M words to be tagged and
 4533 each of the K tags in the tagset. It can be built by composition of $(T_E \circ T_T)$ against an
 4534 unweighted **chain FSA** $M_A(\mathbf{w})$ that is specially constructed to accept only a given
 4535 input w_1, w_2, \dots, w_M , shown in Figure 9.9. The trellis for input \mathbf{w} is built from the
 4536 composition $M_A(\mathbf{w}) \circ (T_E \circ T_T)$. Composing with the unweighted $M_A(\mathbf{w})$ does not
 4537 affect the edge weights from $(T_E \circ T_T)$, but it selects the subset of paths that generate
 4538 the word sequence \mathbf{w} .

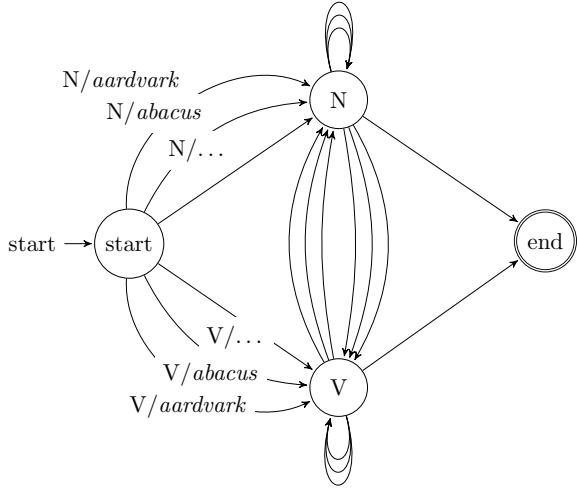


Figure 9.8: Finite state transducer for hidden Markov models, with a small tagset of nouns and verbs. For each pair of tags (including self-loops), there is an edge for every word in the vocabulary. For simplicity, input and output are only shown for the edges from the start state. Weights are also omitted 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 $\log \Pr(Y_m = \diamond \mid Y_{m-1} = i)$.

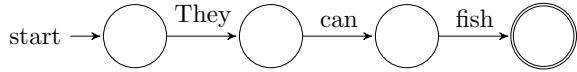


Figure 9.9: Chain finite state acceptor for the input *They can fish*.

4539 9.1.5 *Learning weighted finite state automata

4540 In generative models such as n -gram language models and hidden Markov models, the
 4541 edge weights correspond to log probabilities, which can be obtained from relative fre-
 4542 quency estimation. However, in other cases, we wish to learn the edge weights from in-
 4543 put/output pairs. This is difficult in non-deterministic finite state automata, because we
 4544 do not observe the specific arcs that are traversed in accepting the input, or in transducing
 4545 from input to output. The path through the automaton is a **latent variable**.

4546 Chapter 5 presented one method for learning with latent variables: expectation max-
 4547 imization (EM). This involves computing a distribution $q(\cdot)$ over the latent variable, and
 4548 iterating between updates to this distribution and updates to the parameters — in this
 4549 case, the arc weights. The **forward-backward algorithm** (§ 7.5.3.3) describes a dynamic
 4550 program for computing a distribution over arcs in the trellis structure of a hidden Markov

model, but this is a special case of the more general problem for finite state automata. Eisner (2002) describes an **expectation semiring**, which enables the expected number of transitions across each arc to be computed through a semiring shortest-path algorithm. Alternative approaches for generative models include Markov Chain Monte Carlo (Chiang et al., 2010) and spectral learning (Balle et al., 2011).

Further afield, we can take a perceptron-style approach, with each arc corresponding to a feature. The classic perceptron update would update the weights by subtracting the difference between the feature vector corresponding to the predicted path and the feature vector corresponding to the correct path. Since the path is not observed, we resort to a **hidden variable perceptron**. The model is described formally in § 12.4, but the basic idea is to compute an update from the difference between the features from the predicted path and the features for the best-scoring path that generates the correct output.

9.2 Context-free languages

Beyond the class of regular languages lie the context-free languages. An example of a language that is context-free but not finite state is the set of arithmetic expressions with balanced parentheses. Intuitively, to accept only strings in this language, an FSA would have to “count” the number of left parentheses, and make sure that they are balanced against the number of right parentheses. An arithmetic expression can be arbitrarily long, yet by definition an FSA has a finite number of states. Thus, for any FSA, there will be a string that with too many parentheses to count. More formally, the **pumping lemma** is a proof technique for showing that languages are not regular. It is typically demonstrated for the simpler case $a^n b^n$, the language of strings containing a sequence of a 's, and then an equal-length sequence of b 's.⁴

There are at least two arguments for the relevance of non-regular formal languages to linguistics. First, there are natural language phenomena that are argued to be isomorphic to $a^n b^n$. For English, the classic example is **center embedding**, shown in Figure 9.10. The initial expression *the dog* specifies a single dog. Embedding this expression into *the cat ... chased* specifies a particular cat — the one chased by the dog. This cat can then be embedded again to specify a goat, in the less felicitous but arguably grammatical expression, *the goat the cat the dog chased kissed*, which refers to the goat who was kissed by the cat which was chased by the dog. Chomsky (1957) argues that to be grammatical, a center-embedded construction must be balanced: if it contains n noun phrases (e.g., *the cat*), they must be followed by exactly $n - 1$ verbs. An FSA that could recognize such expressions would also be capable of recognizing the language $a^n b^n$. Because we can prove that no FSA exists for $a^n b^n$, no FSA can exist for center embedded constructions either. En-

⁴Details of the proof can be found in an introductory computer science theory textbook (e.g., Sipser, 2012).

			the dog	
	the cat	the dog	chased	
the goat	the cat	the dog	chased	kissed
			...	

Figure 9.10: Three levels of center embedding

4586 glish includes center embedding, and so the argument goes, English grammar as a whole
 4587 cannot be regular.⁵

4588 A more practical argument for moving beyond regular languages is modularity. Many
 4589 linguistic phenomena — especially in syntax — involve constraints that apply at long
 4590 distance. Consider the problem of determiner-noun number agreement in English: we
 4591 can say *the coffee* and *these coffees*, but not **these coffee*. By itself, this is easy enough to model
 4592 in an FSA. However, fairly complex modifying expressions can be inserted between the
 4593 determiner and the noun:

- 4594 (9.5) the burnt coffee
- 4595 (9.6) the badly-ground coffee
- 4596 (9.7) the burnt and badly-ground Italian coffee
- 4597 (9.8) these burnt and badly-ground Italian coffees
- 4598 (9.9) *these burnt and badly-ground Italian coffee

4599 Again, an FSA can be designed to accept modifying expressions such as *burnt and badly-*
 4600 *ground Italian*. Let's call this FSA F_M . To reject the final example, a finite state acceptor
 4601 must somehow "remember" that the determiner was plural when it reaches the noun *cof-*
 4602 *fee* at the end of the expression. The only way to do this is to make two identical copies
 4603 of F_M : one for singular determiners, and one for plurals. While this is possible in the
 4604 finite state framework, it is inconvenient — especially in languages where more than one
 4605 attribute of the noun is marked by the determiner. **Context-free languages** facilitate mod-
 4606 ularity across such long-range dependencies.

4607 9.2.1 Context-free grammars

4608 Context-free languages are specified by **context-free grammars (CFGs)**, which are tuples
 4609 (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

- 4610 • a finite set of **non-terminals** N ;
- 4611 • a finite alphabet Σ of **terminal symbols**;
- 4612 • a set of **production rules** R , each of the form $A \rightarrow \beta$, where $A \in N$ and $\beta \in (\Sigma \cup N)^*$;
- 4613 • a designated start symbol S .

4614 In the production rule $A \rightarrow \beta$, the left-hand side (LHS) A must be a non-terminal;
 4615 the right-hand side (RHS) can be a sequence of terminals or non-terminals, $\{n, \sigma\}^*, n \in$
 4616 $N, \sigma \in \Sigma$. A non-terminal can appear on the left-hand side of many production rules.
 4617 A non-terminal can appear on both the left-hand side and the right-hand side; this is a
 4618 **recursive production**, and is analogous to self-loops in finite state automata. The name
 4619 “context-free” is based on the property that the production rule depends only on the LHS,
 4620 and not on its ancestors or neighbors; this is analogous to Markov property of finite state
 4621 automata, in which the behavior at each step depends only on the current state, on not on
 4622 the path by which that state was reached.

4623 A **derivation** τ is a sequence of steps from the start symbol S to a surface string $w \in \Sigma^*$,
 4624 which is the **yield** of the derivation. A string w is in a context-free language if there is
 4625 some derivation from S yielding w . **Parsing** is the problem of finding a derivation for a
 4626 string in a grammar. Algorithms for parsing are described in chapter 10.

4627 Like regular expressions, context-free grammars define the language but not the com-
 4628 putation necessary to recognize it. The context-free analogues to finite state acceptors are
 4629 **pushdown automata**, a theoretical model of computation in which input symbols can be
 4630 pushed onto a stack with potentially infinite depth. For more details, see Sipser (2012).

4631 9.2.1.1 Example

4632 Figure 9.11 shows a context-free grammar for arithmetic expressions such as $1 + 2 \div 3 - 4$.
 4633 In this grammar, the terminal symbols include the digits $\{1, 2, \dots, 9\}$ and the op-
 4634 erators $\{+, -, \times, \div\}$. The rules include the $|$ symbol, a notational convenience that makes
 4635 it possible to specify multiple right-hand sides on a single line: the statement $A \rightarrow x | y$

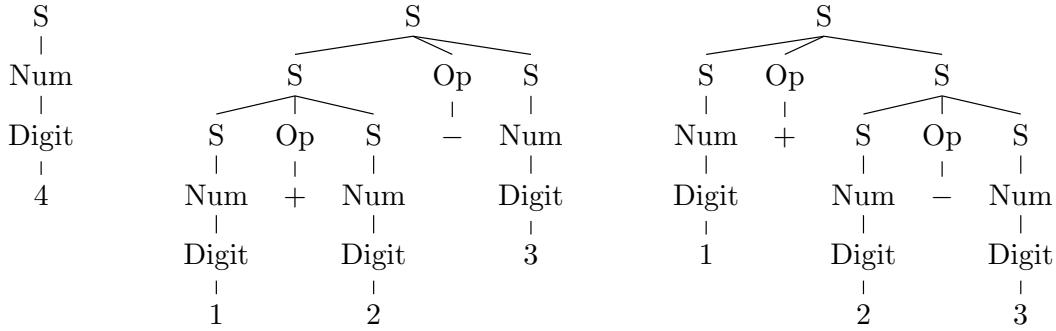


Figure 9.12: Some example derivations from the arithmetic grammar in Figure 9.11

4636 defines *two* productions, $A \rightarrow x$ and $A \rightarrow y$. This grammar is recursive: the non-termals S
4637 and NUM can produce themselves.

4638 Derivations are typically shown as trees, with production rules applied from the top
4639 to the bottom. The tree on the left in Figure 9.12 describes the derivation of a single digit,
4640 through the sequence of productions $S \rightarrow \text{NUM} \rightarrow \text{DIGIT} \rightarrow 4$ (these are all **unary produc-**
4641 **tions**, because the right-hand side contains a single element). The other two trees in
4642 Figure 9.12 show alternative derivations of the string $1 + 2 - 3$. The existence of multiple
4643 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:

$$(S (S (S (\text{Num} (Digit 1))) (\text{Op} +) (S (\text{Num} (Digit 2))))) (\text{Op} -) (S (\text{Num} (Digit 3)))) \quad [9.23]$$

$$(S (S (\text{Num} (Digit 1))) (\text{Op} +) (S (\text{Num} (Digit 2)) (\text{Op} -) (S (\text{Num} (Digit 3)))))). \quad [9.24]$$

4644 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 $n > 0$.

$$\begin{aligned} S &\rightarrow aSb \mid ab \\ S &\rightarrow aSb \mid aabb \mid ab \end{aligned}$$

4645 Two grammars are **weakly equivalent** if they generate the same strings. Two grammars
4646 are **strongly equivalent** if they generate the same strings via the same derivations. The
4647 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:

$$A \rightarrow BC$$

$$A \rightarrow a$$

- 4648 All CFGs can be converted into a CNF grammar that is weakly equivalent. To convert a
 4649 grammar into CNF, we first address productions that have more than two non-terminals
 4650 on the RHS by creating new “dummy” non-terminals. For example, if we have the pro-
 4651 duction,

$$W \rightarrow X Y Z, \quad [9.25]$$

it is replaced with two productions,

$$W \rightarrow X W \setminus X \quad [9.26]$$

$$W \setminus X \rightarrow Y Z. \quad [9.27]$$

- 4652 In these productions, $W \setminus X$ is a new dummy non-terminal. This transformation **binarizes**
 4653 the grammar, which is critical for efficient bottom-up parsing, as we will see in chapter 10.
 4654 Productions whose right-hand side contains a mix of terminal and non-terminal symbols
 4655 can be replaced in a similar fashion.

- 4656 Unary non-terminal productions $A \rightarrow B$ are replaced as follows: identify all produc-
 4657 tions $B \rightarrow \alpha$, and add $A \rightarrow \alpha$ to the grammar. For example, in the grammar described in
 4658 Figure 9.11, we would replace $\text{NUM} \rightarrow \text{DIGIT}$ with $\text{NUM} \rightarrow 1 \mid 2 \mid \dots \mid 9$. However, we
 4659 keep the production $\text{NUM} \rightarrow \text{NUM DIGIT}$, which is a valid binary production.

4660 9.2.2 Natural language syntax as a context-free language

- 4661 Context-free grammars are widely used to represent **syntax**, which is the set of rules that
 4662 determine whether an utterance is judged to be grammatical. If this representation were
 4663 perfectly faithful, then a natural language such as English could be transformed into a
 4664 formal language, consisting of exactly the (infinite) set of strings that would be judged to
 4665 be grammatical by a fluent English speaker. We could then build parsing software that
 4666 would automatically determine if a given utterance were grammatical.⁷

- 4667 Contemporary theories generally do *not* consider natural languages to be context-free
 4668 (see § 9.3), yet context-free grammars are widely used in natural language parsing. The
 4669 reason is that context-free representations strike a good balance: they cover a broad range
 4670 of syntactic phenomena, and they can be parsed efficiently. This section therefore de-
 4671 scribes how to handle a core fragment of English syntax in context-free form, following

⁷You are encouraged to move beyond this cursory treatment of syntax by consulting a textbook on linguistics (e.g., Akmajian et al., 2010; Bender, 2013).

4672 the conventions of the **Penn Treebank** (PTB; Marcus et al., 1993), a large-scale annotation
 4673 of English language syntax. The generalization to “mildly” context-sensitive languages is
 4674 discussed in § 9.3.

4675 The Penn Treebank annotation is a **phrase-structure grammar** of English. This means
 4676 that sentences are broken down into **constituents**, which are contiguous sequences of
 4677 words that function as coherent units for the purpose of linguistic analysis. Constituents
 4678 generally have a few key properties:

4679 **Movement.** Constituents can often be moved around sentences as units.

- 4680 (9.10) Abigail gave (her brother) (a fish).
 4681 (9.11) Abigail gave (a fish) to (her brother).

4682 In contrast, *gave her* and *brother a* cannot easily be moved while preserving gram-
 4683 maticality.

4684 **Substitution.** Constituents can be substituted by other phrases of the same type.

- 4685 (9.12) Max thanked (his older sister).
 4686 (9.13) Max thanked (her).

4687 In contrast, substitution is not possible for other contiguous units like *Max thanked*
 4688 and *thanked his*.

4689 **Coordination.** Coordinators like *and* and *or* can conjoin constituents.

- 4690 (9.14) (Abigail) and (her younger brother) bought a fish.
 4691 (9.15) Abigail (bought a fish) and (gave it to Max).
 4692 (9.16) Abigail (bought) and (greedily ate) a fish.

4693 Units like *brother bought* and *bought a* cannot easily be coordinated.

4694 These examples argue for units such as *her brother* and *bought a fish* to be treated as con-
 4695 stituents. Other sequences of words in these examples, such as *Abigail gave* and *brother*
 4696 *a fish*, cannot be moved, substituted, and coordinated in these ways. In phrase-structure
 4697 grammar, constituents are nested, so that *the senator from New Jersey* contains the con-
 4698 stituent *from New Jersey*, which in turn contains *New Jersey*. The sentence itself is the max-
 4699 imal constituent; each word is a minimal constituent, derived from a unary production
 4700 from a part-of-speech tag. Between part-of-speech tags and sentences are **phrases**. In
 4701 phrase-structure grammar, phrases have a type that is usually determined by their **head**
 4702 **word**: for example, a **noun phrase** corresponds to a noun and the group of words that

4703 modify it, such as *her younger brother*; a **verb phrase** includes the verb and its modifiers,
4704 such as *bought a fish* and *greedily ate it*.

4705 In context-free grammars, each phrase type is a non-terminal, and each constituent is
4706 the substring that the non-terminal yields. Grammar design involves choosing the right
4707 set of non-terminals. Fine-grained non-terminals make it possible to represent more fine-
4708 grained linguistic phenomena. For example, by distinguishing singular and plural noun
4709 phrases, it is possible to have a grammar of English that generates only sentences that
4710 obey subject-verb agreement. However, enforcing subject-verb agreement is considerably
4711 more complicated in languages like Spanish, where the verb must agree in both person
4712 and number with subject. In general, grammar designers must trade off between **over-**
4713 **generation** — a grammar that permits ungrammatical sentences — and **undergeneration**
4714 — a grammar that fails to generate grammatical sentences. Furthermore, if the grammar is
4715 to support manual annotation of syntactic structure, it must be simple enough to annotate
4716 efficiently.

4717 9.2.3 A phrase-structure grammar for English

4718 To better understand how phrase-structure grammar works, let's consider the specific
4719 case of the Penn Treebank grammar of English. The main phrase categories in the Penn
4720 Treebank (PTB) are based on the main part-of-speech classes: noun phrase (NP), verb
4721 phrase (VP), prepositional phrase (PP), adjectival phrase (ADJP), and adverbial phrase
4722 (ADVP). The top-level category is S, which conveniently stands in for both “sentence”
4723 and the “start” symbol. **Complement clauses** (e.g., *I take the good old fashioned ground that*
4724 *the whale is a fish*) are represented by the non-terminal SBAR. The terminal symbols in
4725 the grammar are individual words, which are generated from unary productions from
4726 part-of-speech tags (the PTB tagset is described in § 8.1).

4727 This section explores the productions from the major phrase-level categories, explaining
4728 how to generate individual tag sequences. The production rules are approached in a
4729 “theory-driven” manner: first the syntactic properties of each phrase type are described,
4730 and then some of the necessary production rules are listed. But it is important to keep
4731 in mind that the Penn Treebank was produced in a “data-driven” manner. After the set
4732 of non-terminals was specified, annotators were free to analyze each sentence in what-
4733 ever way seemed most linguistically accurate, subject to some high-level guidelines. The
4734 grammar of the Penn Treebank is simply the set of productions that were required to ana-
4735 lyze the several million words of the corpus. By design, the grammar overgenerates — it
4736 does not exclude ungrammatical sentences.

4737 **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:

$$S \rightarrow ADVP NP VP \quad \text{Unfortunately } Abigail \text{ ate the kimchi.} \quad [9.29]$$

$$S \rightarrow S CC S \quad \text{Abigail ate the kimchi and Max had a burger.} \quad [9.30]$$

$$S \rightarrow VP \quad \text{Eat the kimchi.} \quad [9.31]$$

- 4738 where ADVP is an adverbial phrase (e.g., *unfortunately*, *very unfortunately*) and CC is a
 4739 coordinating conjunction (e.g., *and*, *but*).⁸

4740 **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:

$$NP \rightarrow NN | NNS | NNP | PRP \quad [9.32]$$

$$NP \rightarrow DET NN | DET NNS | DET NNP \quad [9.33]$$

- 4741 The tags NN, NNS, and NNP refer to singular, plural, and proper nouns; PRP refers to
 4742 personal pronouns, and DET refers to determiners. The grammar also contains terminal
 4743 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.

- 4744 These multiple noun constructions can be combined with adjectival phrases and cardinal
 4745 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]

- 4746 These recursive productions are a major source of ambiguity, because the VP and PP non-
 4747 terminals can also generate NP children. Thus, the *the President of the Georgia Institute of*
 4748 *Technology* can be derived in two ways, as can *a whale taken near Shetland in October*.

4749 But aside from these few recursive productions, the noun phrase fragment of the Penn
 4750 Treebank grammar is relatively flat, containing a large of number of productions that go
 4751 from NP directly to a sequence of parts-of-speech. If noun phrases had more internal
 4752 structure, the grammar would need fewer rules, which, as we will see, would make pars-
 4753 ing faster and machine learning easier. Vadas and Curran (2011) propose to add additional
 4754 structure in the form of a new non-terminal called a **nominal modifier** (NML), e.g.,

- 4755 (9.17) (NP (NN crude) (NN oil) (NNS prices)) (PTB analysis)
 4756 (NP (NML (NN crude) (NN oil)) (NNS prices)) (NML-style analysis)

4757 Another proposal is to treat the determiner as the head of a **determiner phrase** (DP;
 4758 Abney, 1987). There are linguistic arguments for and against determiner phrases (e.g.,
 4759 Van Eynde, 2006). From the perspective of context-free grammar, DPs enable more struc-
 4760 tured analyses of some constituents, e.g.,

- 4761 (9.18) (NP (DT the) (JJ white) (NN whale)) (PTB analysis)
 4762 (DP (DT the) (NP (JJ white) (NN whale))) (DP-style analysis).

4763 9.2.3.3 Verb phrases

Verb phrases describe actions, events, and states of being. The PTB tagset distinguishes several classes of verb inflections: base form (VB; *she likes to snack*), present-tense third-person singular (VBD; *she snacks*), present tense but not third-person singular (VBP; *they snack*), past tense (VBD; *they snacked*), present participle (VBG; *they are snacking*), and past participle (VBN; *they had snacked*).⁹ Each of these forms can constitute a verb phrase on its

⁹It bears emphasis the principles governing this tagset design are entirely English-specific: VBP is a meaningful category only because English morphology distinguishes third-person singular from all person-number combinations.

own:

$$VP \rightarrow VB \mid VBZ \mid VBD \mid VBN \mid VBG \mid VBP \quad [9.41]$$

More complex verb phrases can be formed by a number of recursive productions, including the use of coordination, modal verbs (MD; *she should snack*), and the infinitival *to* (TO):

$VP \rightarrow MD \ VP$	<i>She will snack</i>	[9.42]
$VP \rightarrow VBD \ VP$	<i>She had snacked</i>	[9.43]
$VP \rightarrow VBZ \ VP$	<i>She has been snacking</i>	[9.44]
$VP \rightarrow VBN \ VP$	<i>She has been snacking</i>	[9.45]
$VP \rightarrow TO \ VP$	<i>She wants to snack</i>	[9.46]
$VP \rightarrow VP \ CC \ VP$	<i>She buys and eats many snacks</i>	[9.47]

- 4764 Each of these productions uses recursion, with the VP non-terminal appearing in both the
 4765 LHS and RHS. This enables the creation of complex verb phrases, such as *She will have*
 4766 *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]

- 4767 The first production overgenerates, licensing sentences like **Asha sees Boyang eats the kimchi*. This problem could be addressed by designing a more specific set of sentence non-
 4768 terminals, indicating whether the main verb can be conjugated.
 4769

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]

4770 Again, because these productions are not recursive, the grammar must include productions for every verb part-of-speech.

4771 A special set of verbs, known as **copula**, can take **predicative adjectives** as direct objects:

$VP \rightarrow VBZ\ ADJP$ *She is hungry* [9.56]

$VP \rightarrow VBP\ ADJP$ *Success seems increasingly unlikely* [9.57]

4772 The PTB does not have a special non-terminal for copular verbs, so this production generates non-grammatical examples such as **She eats tall*.

4773 **Particles** (PRT as a phrase; RP as a part-of-speech) work to create phrasal verbs:

$VP \rightarrow VB\ PRT$ *She told them to fuck off* [9.58]

$VP \rightarrow VBD\ PRT\ NP$ *They gave up their ill-gotten gains* [9.59]

4774 As the second production shows, particle productions are required for all configurations of verb parts-of-speech and direct objects.

4776 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,

$PP \rightarrow IN\ NP$ *the whiteness of the whale* [9.60]

$PP \rightarrow TO\ NP$ *What the white whale was to Ahab, has been hinted.* [9.61]

Similarly, complement clauses consist of a complementizer (usually a preposition, possibly null) and a sentence,

$SBAR \rightarrow IN\ S$ *She said that it was spicy* [9.62]

$SBAR \rightarrow S$ *She said it was spicy* [9.63]

Adverbial phrases are usually bare adverbs ($ADVP \rightarrow RB$), with a few exceptions:

$ADVP \rightarrow RB\ RBR$ *They went considerably further* [9.64]

$ADVP \rightarrow ADVP\ PP$ *They went considerably further than before* [9.65]

4777 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]

4778 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> <i>or whether they want exports</i>	[9.76]

4779 9.2.4 Grammatical ambiguity and weighted context-free grammars

4780 Context-free parsing is useful not because it determines whether a sentence is grammatical, but mainly because the constituents and their relations can be applied to tasks such as information extraction (chapter 17) and sentence compression (Jing, 2000; Clarke and Lapata, 2008). However, the **ambiguity** of wide-coverage natural language grammars 4781 poses a serious problem for such potential applications. As an example, Figure 9.13 shows 4782 two possible analyses for the simple sentence *We eat sushi with chopsticks*, depending on 4783 whether the *chopsticks* modify *eat* or *sushi*. Realistic grammars can license thousands or 4784 even millions of parses for individual sentences.

Weighted context-free grammars solve this problem by attaching weights to each production. The score of a derivation $\tau = \{(X \rightarrow \alpha)\}$ is the sum of the weights of the productions,

$$\Psi(\tau) = \sum_{(X \rightarrow \alpha) \in \tau} \psi(X \rightarrow \alpha), \quad [9.77]$$

where $\psi(X \rightarrow \alpha)$ is the score of the production $X \rightarrow \alpha$. The parsing problem is then,

$$\hat{\tau} = \underset{\tau: \text{yield}(\tau)=w}{\operatorname{argmax}} \Psi(\tau). \quad [9.78]$$

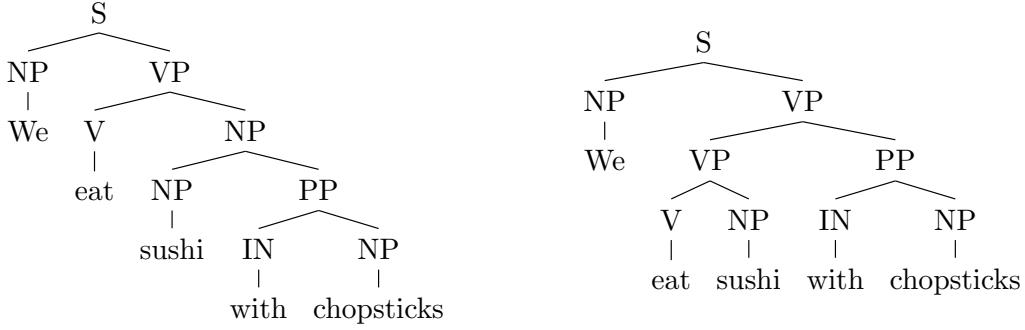


Figure 9.13: Two derivations of the same sentence

4788 Parsing algorithms for weighted CFGs are described in chapter 10; in general, the time
 4789 complexity is cubic in the length of the input. Given a labeled dataset, the weights can be
 4790 set equal to log conditional probabilities, $\psi(X \rightarrow \alpha) = \log p(\alpha | X)$. Setting the weights
 4791 in this way defines a **probabilistic context-free grammar** (PCFG), in which the total score
 4792 of a derivation is equal to the log of the joint probability, $\log p(w, \tau)$. Alternatively, the
 4793 weights can be learned using discriminative algorithms such as perceptron. For more
 4794 details, see § 10.3.

4795 9.3 *Mildly context-sensitive languages

4796 Beyond context-free languages lie **context-sensitive languages**, in which the expansion
 4797 of a non-terminal depends on its neighbors. In the general class of context-sensitive
 4798 languages, computation becomes much more challenging: the membership problem for
 4799 context-sensitive languages is PSPACE-complete. Since PSPACE contains the complexity
 4800 class NP (problems that can be solved in polynomial time on a non-deterministic Turing
 4801 machine), PSPACE-complete problems cannot be solved efficiently if $P \neq NP$. Thus, de-
 4802 signing an efficient parsing algorithm for the full class of context-sensitive languages is
 4803 probably hopeless.¹⁰

4804 However, Joshi (1985) identifies a set of properties that define **mildly context-sensitive**
 4805 **languages**, which are a strict subset of context-sensitive languages. Like context-free lan-
 4806 guages, mildly context-sensitive languages are efficiently parseable. However, the mildly
 4807 context-sensitive languages include non-context-free languages, such as the “copy lan-
 4808 guage” $\{ww \mid w \in \Sigma^*\}$ and the language $a^m b^n c^m d^n$. Both are characterized by **cross-**

¹⁰If $PSPACE \neq NP$, 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).

4809 **serial dependencies**, linking symbols at long distance across the string.¹¹ For example, in
 4810 the language $a^n b^m c^n d^m$, each a symbol is linked to exactly one c symbol, regardless of the
 4811 number of intervening b symbols.

4812 9.3.1 Context-sensitive phenomena in natural language

4813 Such phenomena are occasionally relevant to natural language. A classic example is found
 4814 in Swiss-German (Shieber, 1985), in which sentences such as *we let the children help Hans*
 4815 *paint the house* are realized by listing all nouns before all verbs, i.e., *we the children Hans the*
 4816 *house let help paint*. Furthermore, each noun's determiner is dictated by the noun's **case**
 4817 **marking** (the role it plays with respect to the verb). Using an argument that is analogous
 4818 to the earlier discussion of center-embedding (§ 9.2), Shieber argues that these case marking
 4819 constraints are a cross-serial dependency, homomorphic to $a^m b^n c^m d^m$, and therefore
 4820 not context-free.

4821 As with the move from regular to context-free languages, mildly context-sensitive lan-
 4822 guages can be motivated by expedience. While infinite sequences of cross-serial depen-
 4823 dencies cannot be handled by context-free grammars, even finite sequences of cross-serial
 4824 dependencies are more convenient to handle using a mildly context-sensitive formalism
 4825 like **tree-adjoining grammar** (TAG) and **combinatory categorial grammar** (CCG). Fur-
 4826 thermore, TAG-inspired parsers have been shown to be particularly effective in parsing
 4827 the Penn Treebank (Collins, 1997; Carreras et al., 2008), and CCG plays a leading role in
 4828 current research on semantic parsing (Zettlemoyer and Collins, 2005). Furthermore, these
 4829 two formalisms are weakly equivalent: any language that can be specified in TAG can also
 4830 be specified in CCG, and vice versa (Joshi et al., 1991). The remainder of the chapter gives
 4831 a brief overview of CCG, but you are encouraged to consult Joshi and Schabes (1997) and
 4832 Steedman and Baldridge (2011) for more detail on TAG and CCG respectively.

4833 9.3.2 Combinatory categorial grammar

4834 In combinatory categorial grammar, structural analyses are built up through a small set
 4835 of generic combinatorial operations, which apply to immediately adjacent sub-structures.
 4836 These operations act on the categories of the sub-structures, producing a new structure
 4837 with a new category. The basic categories include S (sentence), NP (noun phrase), VP
 4838 (verb phrase) and N (noun). The goal is to label the entire span of text as a sentence, S.

4839 Complex categories, or types, are constructed from the basic categories, parentheses,
 4840 and forward and backward slashes: for example, S/NP is a complex type, indicating a
 4841 sentence that is lacking a noun phrase to its right; S\NP is a sentence lacking a noun

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 $\{a^{2^n} \mid n \geq 0\}$.

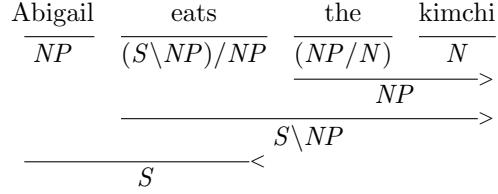


Figure 9.14: A syntactic analysis in CCG involving forward and backward function application

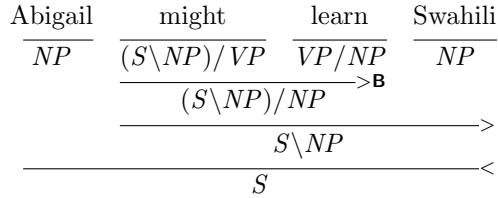


Figure 9.15: A syntactic analysis in CCG involving function composition (example modified from Steedman and Baldridge, 2011)

4842 phrase to its left. Complex types act as functions, and the most basic combinatory oper-
 4843 ations are function application to either the right or left neighbor. For example, the type
 4844 of a verb phrase, such as *eats*, would be $S\backslash NP$. Applying this function to a subject noun
 4845 phrase to its left results in an analysis of *Abigail eats* as category S , indicating a successful
 4846 parse.

4847 Transitive verbs must first be applied to the direct object, which in English appears to
 4848 the right of the verb, before the subject, which appears on the left. They therefore have the
 4849 more complex type $(S\backslash NP)/NP$. Similarly, the application of a determiner to the noun at
 4850 its right results in a noun phrase, so determiners have the type NP/N . Figure 9.14 pro-
 4851 vides an example involving a transitive verb and a determiner. A key point from this
 4852 example is that it can be trivially transformed into phrase-structure tree, by treating each
 4853 function application as a constituent phrase. Indeed, when CCG’s only combinatory op-
 4854 erators are forward and backward function application, it is equivalent to context-free
 4855 grammar. However, the location of the “effort” has changed. Rather than designing good
 4856 productions, the grammar designer must focus on the **lexicon** — choosing the right cate-
 4857 gories for each word. This makes it possible to parse a wide range of sentences using only
 4858 a few generic combinatory operators.

4859 Things become more interesting with the introduction of two additional operators:
 4860 **composition** and **type-raising**. Function composition enables the combination of com-
 4861 plex types: $X/Y \circ Y/Z \Rightarrow_B X/Z$ (forward composition) and $Y\backslash Z \circ X\backslash Y \Rightarrow_B X\backslash Z$ (back-

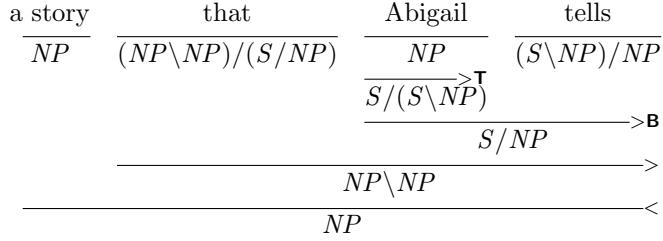


Figure 9.16: A syntactic analysis in CCG involving an object relative clause (based on slides from Alex Clark)

ward composition).¹² Composition makes it possible to “look inside” complex types, and combine two adjacent units if the “input” for one is the “output” for the other. Figure 9.15 shows how function composition can be used to handle modal verbs. While this sentence can be parsed using only function application, the composition-based analysis is preferable because the unit *might learn* functions just like a transitive verb, as in the example *Abigail studies Swahili*. This in turn makes it possible to analyze conjunctions such as *Abigail studies and might learn Swahili*, attaching the direct object *Swahili* to the entire conjoined verb phrase *studies and might learn*. The Penn Treebank grammar fragment from § 9.2.3 would be unable to handle this case correctly: the direct object *Swahili* could attach only to the second verb *learn*.

Type raising converts an element of type X to a more complex type: $X \Rightarrow_T T/(T\backslash X)$ (forward type-raising to type T), and $X \Rightarrow_T T\backslash(T/X)$ (backward type-raising to type T). Type-raising makes it possible to reverse the relationship between a function and its argument — by transforming the argument into a function over functions over arguments! An example may help. Figure 9.15 shows how to analyze an object relative clause, *a story that Abigail tells*. The problem is that *tells* is a transitive verb, expecting a direct object to its right. As a result, *Abigail tells* is not a valid constituent. The issue is resolved by raising *Abigail* from NP to the complex type $(S/NP)\backslash NP$. This function can then be combined with the transitive verb *tells* by forward composition, resulting in the type (S/NP) , which is a sentence lacking a direct object to its right.¹³ From here, we need only design the lexical entry for the complementizer *that* to expect a right neighbor of type (S/NP) , and the remainder of the derivation can proceed by function application.

Composition and type-raising give CCG considerable power and flexibility, but at a price. The simple sentence *Abigail tells Max* can be parsed in two different ways: by function application (first forming the verb phrase *tells Max*), and by type-raising and composition (first forming the non-constituent *Abigail tells*). This **derivational ambiguity** does

¹²The subscript **B** follows notation from Curry and Feys (1958).

¹³The missing direct object would be analyzed as a **trace** in CFG-like approaches to syntax, including the Penn Treebank.

not affect the resulting linguistic analysis, so it is sometimes known as **spurious ambiguity**. Hockenmaier and Steedman (2007) present a translation algorithm for converting the Penn Treebank into CCG derivations, using composition and type-raising only when necessary.

Exercises

1. Sketch out the state diagram for finite-state acceptors for the following languages on the alphabet $\{a, b\}$.

- a) Even-length strings. (Be sure to include 0 as an even number.)
- b) Strings that contain aaa as a substring.
- c) Strings containing an even number of a and an odd number of b symbols.
- d) Strings in which the substring bbb must be terminal if it appears — the string need not contain bbb , but if it does, nothing can come after it.

2. Levenshtein edit distance is the number of insertions, substitutions, or deletions required to convert one string to another.

- a) Define a finite-state acceptor that accepts all strings with edit distance 1 from the target string, *target*.
- b) Now think about how to generalize your design to accept all strings with edit distance from the target string equal to d . If the target string has length ℓ , what is the minimal number of states required?

3. Construct an FSA in the style of Figure 9.3, which handles the following examples:

- *nation*/N, *national*/ADJ, *nationalize*/V, *nationalizer*/N
- *America*/N, *American*/ADJ, *Americanize*/V, *Americanizer*/N

Be sure that your FSA does not accept any further derivations, such as **nationalizeral* and **Americanizern*.

- 4. Show how to construct a trigram language model in a weighted finite-state acceptor. Make sure that you handle the edge cases at the beginning and end of the sequence accurately.
- 5. Extend the FST in Figure 9.6 to handle the other two parts of rule 1a of the Porter stemmer: $-sses \rightarrow ss$, and $-ies \rightarrow -i$.

4917 6. § 9.1.4.4 describes T_O , a transducer that captures English orthography by transduc-
 4918 ing $\text{cook} + \text{ed} \rightarrow \text{cooked}$ and $\text{bake} + \text{ed} \rightarrow \text{baked}$. Design an unweighted finite-state
 4919 transducer that captures this property of English orthography.

4920 Next, augment the transducer to appropriately model the suffix $-s$ when applied to
 4921 words ending in s , e.g. $\text{kiss} + s \rightarrow \text{kisses}$.

4922 7. Add parenthesization to the grammar in Figure 9.11 so that it is no longer ambigu-
 4923 ous.

4924 8. Construct three examples — a noun phrase, a verb phrase, and a sentence — which
 4925 can be derived from the Penn Treebank grammar fragment in § 9.2.3, yet are not
 4926 grammatical. Avoid reusing examples from the text. Optionally, propose corrections
 4927 to the grammar to avoid generating these cases.

4928 9. Produce parses for the following sentences, using the Penn Treebank grammar frag-
 4929 ment from § 9.2.3.

4930 (9.19) This aggression will not stand.

4931 (9.20) I can get you a toe.

4932 (9.21) Sometimes you eat the bar and sometimes the bar eats you.

4933 Then produce parses for three short sentences from a news article from this week.

4934 10. * One advantage of CCG is its flexibility in handling coordination:

4935 (9.22) *Abigail and Max speak Swahili*

4936 (9.23) *Abigail speaks and Max understands Swahili*

Define the lexical entry for *and* as

$$\text{and} := (X/X) \setminus X, \quad [9.79]$$

4937 where X can refer to any type. Using this lexical entry, show how to parse the two
 4938 examples above. In the second example, *Swahili* should be combined with the coor-
 4939 dination *Abigail speaks and Max understands*, and not just with the verb *understands*.

4940 **Chapter 10**

4941 **Context-free Parsing**

4942 Parsing is the task of determining whether a string can be derived from a given context-
4943 free grammar, and if so, how. The parse of a sentence is a hierarchical structure of nested
4944 constituents. This structure helps to answer the basic questions of who-did-what-to-
4945 whom, and is useful for various downstream tasks, such as semantic analysis and in-
4946 formation extraction.

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.,

$$(((w_1 w_2) w_3) w_4) w_5), \quad (((w_1 (w_2 w_3)) w_4) w_5), \quad ((w_1 (w_2 (w_3 w_4))) w_5), \quad \dots$$

4947 The number of binary bracketings is a **Catalan number**, which grows super-exponentially
4948 in the length of the sentence, $C_n = \frac{(2n)!}{(n+1)!n!}$. As with sequence labeling, it is only possi-
4949 ble to search the space of possible derivations by resorting to independence assumptions,
4950 which allow us to search efficiently by reusing shared substructures with dynamic pro-
4951 gramming. This chapter will focus on a bottom-up dynamic programming algorithm
4952 called **CKY**. CKY enables exhaustive search of the space of possible parses, but imposes
4953 strict limitations on the form of scoring function. These limitations can be relaxed by
4954 abandoning exhaustive search. A few non-exact search methods will be discussed at the
4955 end of this chapter, and one of them — **transition-based parsing** — will be the focus of
4956 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

4957 10.1 Deterministic bottom-up parsing

4958 The **CKY algorithm**¹ is a bottom-up approach to parsing in a context-free grammar. It
 4959 efficiently tests whether a string is in a language, without considering all possible parses.
 4960 The algorithm first forms small constituents, and then tries to merge them into larger
 4961 constituents.

4962 To understand the algorithm, consider the input, *we eat sushi with chopsticks*. According
 4963 to the toy grammar in Table 10.1, each terminal symbol can be generated by exactly one
 4964 unary production, resulting in the sequence NP V NP IN NP. Next, we can try to apply
 4965 binary productions to merge adjacent symbols into larger constituents: we could merge V
 4966 NP into VP, or we could merge IN NP into PP. The goal of bottom-up parsing is to find
 4967 some series of mergers that ultimately results in the start symbol S covering the entire
 4968 input.

4969 The CKY algorithm systematizes this approach, incrementally constructing a table (or
 4970 **chart**) t in which each cell $t[i, j]$ contains the set of nonterminals that can derive the span
 4971 $w_{i+1:j}$. The algorithm fills in the upper right triangle of the table; it begins with the di-
 4972 agonal, which corresponds to substrings of length 1, and the computes derivations for
 4973 progressively larger substrings, until reaching the upper right corner $t[0, M]$, which cor-
 4974 responds to the entire input. If the start symbol S is in $t[0, M]$, then the string w is in
 4975 the language defined by the grammar. This process is detailed in Algorithm 13, and the
 4976 resulting data structure is shown in Figure 10.1. Informally, here's how it works:

- 4977 • Begin by filling in the diagonal: the entries $t[m - 1, m]$ for all $m \in \{1 \dots M\}$. These
 4978 are filled with terminal productions that yield the individual tokens; for the word
 4979 $w_2 = \text{sushi}$, we fill in $t[1, 2] = \{\text{NP}\}$, and so on.
- 4980 • Then fill in the next diagonal, in which each cell corresponds to a subsequence of
 4981 length two: $t[0, 2], t[1, 3], \dots, t[M - 2, M]$. These cells are filled in by looking for
 4982 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 sometimes called **chart parsing**, because of its 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  $X \rightarrow (\text{TRACEBACK}(Y, i, k, t), \text{TRACEBACK}(Z, k, j, t))$ 

```

4983 sponding to left and right children. For example, the cell $t[1, 3]$ includes VP because
 4984 the grammar includes the production $\text{VP} \rightarrow \text{V NP}$, and the chart contains $\text{V} \in t[1, 2]$
 4985 and $\text{NP} \in t[2, 3]$.

- 4986 • At the next diagonal, the entries correspond to spans of length three. At this level,
 4987 there is an additional decision to make: where to split the left and right children.
 4988 The cell $t[i, j]$ corresponds to the subsequence $w_{i+1:j}$, and we must choose some
 4989 *split point* $i < k < j$, so that $w_{i+1:k}$ is the left child and $w_{k+1:j}$ is the right child.
 4990 We do this by considering all possible k , and looking for productions that generate
 4991 elements in $t[i, k]$ and $t[k, j]$; the left-hand side of all such productions can be added
 4992 to $t[i, j]$. When it is time to compute $t[i, j]$, the cells $t[i, k]$ and $t[k, j]$ have already
 4993 been filled in, since these cells correspond to shorter sub-strings of the input.
- 4994 • The process continues until we reach $t[0, M]$.

4995 Figure 10.1 shows the chart that arises from parsing the sentence *we eat sushi with chopsticks*
 4996 using the grammar defined above.

4997 **10.1.1 Recovering the parse tree**

4998 As with the Viterbi algorithm, it is possible to identify a successful parse by storing and
 4999 traversing an additional table of back-pointers. If we add an entry X to cell $t[i, j]$ by using
 5000 the production $X \rightarrow YZ$ and the split point k , then we keep back-pointers $b[i, j, X] =$
 5001 (Y, Z, k) . Once the table is constructed, we start with the back-pointers in $b[0, M, S]$, and
 5002 recursively follow them until they ground out at terminal productions.

5003 For ambiguous sentences, there will be multiple paths to reach $S \in t[0, M]$. For ex-
 5004 ample, in Figure 10.1, the goal state $S \in t[0, M]$ is reached through the state $VP \in t[1, 5]$,
 5005 and there are two different ways to generate this constituent: one with *(eat sushi)* and
 5006 *(with chopsticks)* as children, and another with *(eat)* and *(sushi with chopsticks)* as children.
 5007 The presence of multiple paths indicates that the input could have been generated by the
 5008 grammar in more than one way. In Algorithm 13, one of these derivations is selected arbi-
 5009 trarily. § 10.3 describes how weighted context-free grammars can select the optimal parse,
 5010 according to a scoring function.

5011 **10.1.2 Non-binary productions**

5012 The CKY algorithm assumes that all productions with non-terminals on the right-hand
 5013 side (RHS) are binary. But in real grammars, such as the one considered in chapter 9,
 5014 there will be productions with more than two elements on the right-hand side, and other
 5015 productions with only a single element.

- 5016 • For productions with more than two elements on the right-hand side, we **binarize**,
 5017 creating additional non-terminals (see § 9.2.1.2). For example, if we have the
 5018 production $VP \rightarrow V NP NP$ (for ditransitive verbs), we might convert to $VP \rightarrow$
 5019 $VP_{ditrans}/NP NP$, and then add the production $VP_{ditrans}/NP \rightarrow V NP$.
- 5020 • What about unary productions like $VP \rightarrow V$? In practice, this is handled by mak-
 5021 ing a second pass on each diagonal, in which each cell $t[i, j]$ is augmented with all
 5022 possible unary productions capable of generating each item already in the cell —
 5023 formally, $t[i, j]$ is extended to its **unary closure**. Suppose the example grammar in
 5024 Table 10.1 were extended to include the production $VP \rightarrow V$, enabling sentences
 5025 with intransitive verb phrases, like *we eat*. Then the cell $t[1, 2]$ — corresponding to
 5026 the word *eat* — would first include the set $\{V\}$, and would be augmented to the set
 5027 $\{V, VP\}$ during this second pass.

5028 **10.1.3 Complexity**

5029 For an input of length M and a grammar with R productions and N non-terminals, the
 5030 space complexity of the CKY algorithm is $\mathcal{O}(M^2N)$: the number of cells in the chart is
 5031 $\mathcal{O}(M^2)$, and each cell must hold $\mathcal{O}(N)$ elements. The time complexity is $\mathcal{O}(M^3R)$. Each

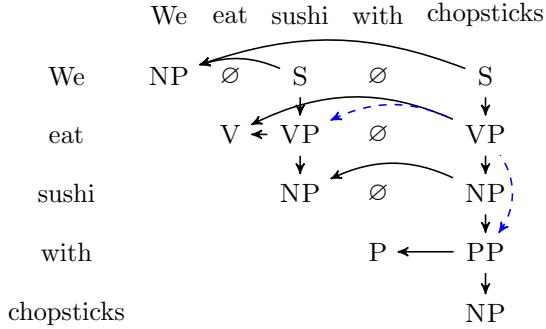


Figure 10.1: An example completed CKY chart. The solid and dashed lines show the back pointers resulting from the two different derivations of VP in position $t[1, 5]$.

5032 cell is computed by searching over $\mathcal{O}(M)$ split points, with R possible productions for
 5033 each split point. Both the time and space complexity are considerably worse than the
 5034 finite-state algorithms of Viterbi and forward-backward, which are linear in the length of
 5035 the input.

5036 10.2 Ambiguity

5037 Syntactic ambiguity is endemic to natural language. Here are a few broad categories:

- 5038 • **Attachment ambiguity:** e.g., *we eat sushi with chopsticks*, *I shot an elephant in my pajamas*. In each of these examples, the prepositions (*with*, *in*) can attach to either the
 5039 verb or the direct object.
- 5041 • **Modifier scope:** e.g., *southern food store*, *plastic cup holder*. In these examples, the first
 5042 word could be modifying the subsequent adjective, or the final noun.
- 5043 • **Particle versus preposition:** e.g., *the puppy tore up the staircase*. Phrasal verbs like
 5044 *tore up* often include particles which could also act as prepositions.
- 5045 • **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
 5046 object (*the professor*).
- 5049 • **Coordination scope:** e.g., *"I see," said the blind man, as he picked up the hammer and
 5050 saw*. In this example, the lexical ambiguity for *saw* enables it to be coordinated either
 5051 with the noun *hammer* or the verb *picked up*.

5052 These forms of ambiguity can combine, so that seemingly simple headlines like *Fed*
 5053 *raises interest rates* have dozens of possible analyses, even in a minimal grammar. Broad

coverage grammars permit millions of parses of typical sentences. While careful grammar design can chip away at this ambiguity, a better strategy is 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 brackets in the system parse that match a bracket in the reference parse.

Recall: the fraction of brackets in the reference parse that match a bracket in the system parse.

In **labeled precision** and **recall**, the system must also match the non-terminals for each bracket; in **unlabeled precision** and **recall**, it is only required to match the bracketing 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.

The labeled and unlabeled precision of this parse is $\frac{3}{4} = 0.75$, and the recall is $\frac{3}{4} = 0.75$, for an F-measure of 0.75. For an example in which precision and recall are not equal, suppose the reference parse instead included the production $VP \rightarrow V NP PP$. In this parse, the reference does not contain the constituent $w_{2:3}$, so the recall would be 1.

²Most parsing papers do not report results on this metric, but Finkel et al. (2008) find that a near-state-of-the-art 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.[[todo: update](#)]

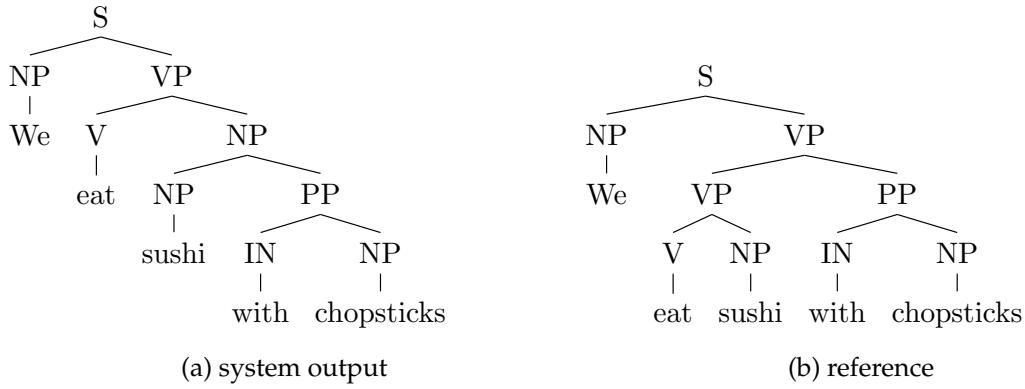


Figure 10.2: Two possible analyses from the grammar in Table 10.1

5085 10.2.2 Local solutions

5086 Some ambiguity can be resolved locally. Consider the following examples,

5087 (10.1) We met the President on Monday.

5088 (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*. To resolve this ambiguity, we can use a labeled corpus to compute the likelihood of observing the preposition alongside each candidate attachment point,

$$p(on \mid met), \quad p(on \mid President) \quad [10.1]$$

$$p(of \mid met), \quad p(of \mid President). \quad [10.2]$$

5089 A comparison of these probabilities would successfully resolve this case (Hindle and
5090 Rooth, 1993). Other cases, such as the example ...*eat sushi with chopsticks*, require con-
5091 sidering the object of the preposition as well. With sufficient labeled data, the problem of
5092 prepositional phrase attachment can be treated as a text classification task (Ratnaparkhi
5093 et al., 1994).

5094 However, there are inherent limitations to local solutions. While toy examples may
5095 have just a few ambiguities to resolve, realistic sentences have thousands or millions of
5096 possible parses. Furthermore, attachment decisions are interdependent, as shown in the
5097 following garden path example:

5098 (10.3) *Cats scratch people with claws with knives.*

5099 We may want to attach *with claws* to *scratch*, as would be correct in the shorter sentence
5100 in *cats scratch people with claws*. But this leaves nowhere to attach *with knives*. The correct

interpretation can be identified only by considering the attachment decisions jointly. The huge number of potential parses may seem to make exhaustive search impossible. But as with sequence labeling, we can use independence assumptions to search this space efficiently.

10.3 Weighted Context-Free Grammars

In a **weighted context-free grammar** (WCFG), each production $X \rightarrow \alpha$ is associated with a score $\psi(X \rightarrow \alpha)$. The score of a derivation is the combination of the scores of all the productions:

$$\Psi(\tau) = \bigotimes_{(X \rightarrow \alpha, (i, j, k)) \in \tau} \psi(X \rightarrow \alpha), \quad [10.3]$$

where τ is regarded as a set of **anchored productions** $\tau = \{X \rightarrow \alpha, (i, j, k)\}$, with X corresponding to the left-hand side non-terminal, α corresponding to the right-hand side; for grammars in Chomsky normal form, α is either a pair of non-terminals or a terminal symbol. The tuple (i, j, k) anchors the production in the input, with X deriving the span $w_{i+1:j}$. For binary productions, $w_{i+1:k}$ indicates the span of the left child, and $w_{k+1:j}$ indicates the span of the right child; for unary productions, k is ignored. In a weighted context-free grammar, the anchors are not considered in the production scores $\psi(X \rightarrow \alpha)$, but they are required to link together the productions into an analysis of the input.

The production scores are combined with the semiring operator \bigotimes (see § 7.7.3), which unifies several types of weighted context free grammars as special cases.

- When each $\psi(X \rightarrow \alpha)$ is a *conditional probability* $p(\alpha | X)$, then \bigotimes corresponds to multiplication, so that $\Psi(\tau)$ is the probability of the full derivation. This special case is explored in § 10.3.2.
- When $\psi(X \rightarrow \alpha)$ is a *conditional log-probability*, \bigotimes corresponds to addition, so that $\Psi(\tau)$ is the log probability of the full derivation. We also set \bigotimes to addition when each $\psi(X \rightarrow \alpha)$ is the output of a discriminatively-trained scoring function, such as $\psi(X \rightarrow \alpha) = \theta \cdot f(X, \alpha, w)$.
- When $\psi(X \rightarrow \alpha)$ is a *Boolean truth value* $\{\top, \perp\}$, then \bigotimes is logical conjunction. This is equivalent to the unweighted context-free grammar considered in the previous section.

Regardless of how the scores are combined, the key point is the independence assumption: the score for a derivation is the combination of the independent scores for each production, and these scores do not depend on any other part of the derivation. For example, if two non-terminals are siblings, the scores of productions from these non-terminals

		$\psi(\cdot)$	$\exp \psi(\cdot)$
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.

5133 are computed independently. This independence assumption is analogous to the first-
 5134 order Markov assumption in sequence labeling, where the score for transitions between
 5135 tags depends only on the previous tag and current tag, and not on the history. As with
 5136 sequence labeling, this assumption makes it possible to find the optimal parse efficiently;
 5137 its linguistic limitations are explored in § 10.5.

Example Consider the weighted grammar shown in Table 10.2, and the analysis in Figure 10.2b. Since the weights are not probabilities, we set $\otimes = +$, obtaining the following score:

$$\begin{aligned} \Psi(\tau) &= \psi(S \rightarrow \text{NP VP}) + \psi(VP \rightarrow \text{VP PP}) + \psi(VP \rightarrow \text{V NP}) + \psi(PP \rightarrow \text{IN NP}) \\ &\quad + \psi(NP \rightarrow \text{we}) + \psi(V \rightarrow \text{eat}) + \psi(NP \rightarrow \text{sushi}) + \psi(IN \rightarrow \text{with}) + \psi(NP \rightarrow \text{chopsticks}) \end{aligned} \quad [10.4]$$

$$= 0 - 2 - 1 + 0 - 2 + 0 - 3 + 0 - 3 = -11. \quad [10.5]$$

5138 In the alternative parse in Figure 10.2a, the production $VP \rightarrow \text{VP PP}$ (with score -2) is
 5139 replaced with the production $NP \rightarrow \text{NP PP}$ (with score -1); all other productions are the
 5140 same. As a result, the score for this parse is -10.

5141 This example hints at a big problem with WCFG parsing on non-terminals such as
 5142 NP, VP, and PP: a WCFG will *always* prefer either VP or NP attachment, without regard
 5143 to what is being attached! This problem is addressed in § 10.5.

5144 **10.3.1 Parsing with weighted context-free grammars**

In a weighted context-free grammar, the task of parsing is to identify the best-scoring derivation,

$$\hat{\tau} = \operatorname{argmax}_{\tau: \text{yield}_G(\tau) = w} \bigotimes_{(X \rightarrow \alpha, (i, j, k)) \in \tau} \psi(X \rightarrow \alpha),$$

5145 The set $\{\tau : w = \text{yield}_G(\tau)\}$ is the set of all possible derivations of w in a weighted
5146 context-free grammar G .

5147 This optimization problem can be solved by modifying the CKY algorithm. In the
5148 deterministic CKY algorithm, each cell $t[i, j]$ stored a set of non-terminals capable of de-
5149 riving the span $w_{i+1:j}$. We now augment the table so that the cell $t[i, j, X]$ is the *score of the*
5150 *best-scoring derivation* of $w_{i+1:j}$ from non-terminal X . This score is computed recursively:
5151 for anchored binary production $(X \rightarrow Y Z, (i, j, k))$, we compute:

- 5152 • the score of the production, $\psi(X \rightarrow Y Z)$;
- 5153 • the score of the left child, $t[i, k, Y]$;
- 5154 • the score of the right child, $t[k, j, Z]$.

5155 These scores are combined by semiring multiplication \otimes . As in the unscored CKY algo-
5156 rithm, the table is constructed by considering spans of increasing length, so the scores
5157 for spans $t[i, k, Y]$ and $t[k, j, Z]$ are guaranteed to be available at the time we compute
5158 the score $t[i, j, X]$. The value $t[0, M, S]$ is the score of the best derivation of w from the
5159 grammar. Algorithm 14 formalizes this procedure.

5160 As in unweighted CKY, the parse is recovered from the table of back pointers b , where
5161 each $b[i, j, X]$ stores the argmax split point k and production $X \rightarrow Y Z$ in the derivation of
5162 $w_{i+1:j}$ from X . The best parse can be obtained by tracing these pointers backwards from
5163 $b[0, M, S]$, all the way to the terminal symbols. This is analogous to the computation of the
5164 best sequence of labels in the Viterbi algorithm by tracing pointers backwards from the
5165 end of the trellis. Note that we need only store back-pointers for the *best* path to $t[i, j, X]$;
5166 this follows from the locality assumption that the global score for a parse is a combination
5167 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 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. In the example, each of these spans is unambiguous. Things get interesting when

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
     $t[i, j, X] \leftarrow \bar{1}$ 
     $b[i, j, X] \leftarrow \emptyset$ 
  for  $m \in \{1, 2, \dots, M\}$  do
    for all  $X \in N$  do
       $t[m, m + 1, X] \leftarrow \psi(X \rightarrow w_m)$ 
    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
           $t[m, m + \ell, X] \leftarrow \max_{k, Y, Z} \psi(X \rightarrow Y Z) \otimes t[m, k, Y] \otimes t[k, m + \ell, Z]$ 
           $b[m, m + \ell, X] \leftarrow \operatorname{argmax}_{k, Y, Z} \psi(X \rightarrow Y Z) \otimes t[m, k, Y] \otimes t[k, m + \ell, Z]$ 
    return TRACEBACK( $S, 0, M, b$ )

```

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,

$$\begin{aligned} t[1, 5, \text{VP}] &= \max(\psi(\text{VP} \rightarrow \text{VP PP}) + t[1, 3, \text{VP}] + t[3, 5, \text{PP}], \\ &\quad \psi(\text{VP} \rightarrow \text{V NP}) + t[1, 2, \text{V}] + t[2, 5, \text{NP}]) \end{aligned} \quad [10.6]$$

$$= \max(-2 - 4 - 3, -1 + 0 - 7) = -8. \quad [10.7]$$

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.

10.3.2 Probabilistic context-free grammars

Probabilistic context-free grammars (PCFGs) are a special case of weighted context-free grammars that arises when the weights correspond to probabilities. Specifically, the weight $\psi(X \rightarrow \alpha) = p(\alpha | X)$, where the probability of the right-hand side α is conditioned on the non-terminal X . These probabilities must be normalized over all possible right-hand sides, so that $\sum_{\alpha} p(\alpha | X) = 1$, for all X . For a given parse τ , the product of the probabilities of the productions is equal to $p(\tau)$, under the **generative model** $\tau \sim \text{DRAWSUBTREE}(S)$, where the function DRAWSUBTREE is defined in Algorithm 15.

The conditional probability of a parse given a string is,

$$p(\tau | w) = \frac{p(\tau)}{\sum_{\tau': \text{yield}(\tau')=w} p(\tau')}. \quad [10.8]$$

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

```

5178 The numerator $p(\tau)$ can be computed by the CKY algorithm for weighted context-free
 5179 grammars, where $\otimes = \times$. As a result, the CKY algorithm can identify $\hat{\tau} = \operatorname{argmax}_{\tau} p(\tau)$,
 5180 without computing the normalization term in the denominator. If a normalized probability
 5181 $p(\tau | w)$ is required, the denominator of Equation 10.8 can be computed by the **inside**
 5182 **recurrence**, described 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 that yield the string *we eat sushi with chopsticks*, $\Psi(\tau_1) = \log p(\tau_1) = -10$ and $\Psi(\tau_2) = \log p(\tau_2) = -11$. Therefore, the conditional probability $p(\tau_1 | w)$ is equal to,

$$p(\tau_1 | w) = \frac{p(\tau_1)}{p(\tau_1) + p(\tau_2)} = \frac{\exp \Psi(\tau_1)}{\exp \Psi(\tau_1) + \exp \Psi(\tau_2)} = \frac{2^{-10}}{2^{-10} + 2^{-11}} = \frac{2}{3}. \quad [10.9]$$

5183 **The inside recurrence** The denominator of Equation 10.8 can be viewed as a language
 5184 model, summing over all parses that yield the string w ,

$$p(w) = \sum_{\tau': \text{yield } \tau' = w} p(\tau'). \quad [10.10]$$

Just as the CKY algorithm makes it possible to maximize over all such analyses, with a slight modification it can also compute their sum. The only change that is required is to replace the maximization over split points k and productions $X \rightarrow Y Z$ with a sum. The two algorithms can be written in a single general form using the semiring addition operator \oplus : to find the best parse, we set \oplus to maximization; to compute the sum of parses, we set \oplus to addition. Thus, the cell $t[i, j, X]$ now contains the total probability of

deriving $w_{i+1:j}$ from X ,

$$t[i, j, X] = \bigoplus_{k, Y, Z} \psi(X \rightarrow Y Z) \otimes t[i, k, Y] \otimes t[k, j, Z] \quad [10.11]$$

$$= \sum_{k, Y, Z} p(Y Z | X) \times p(Y \rightarrow w_{i+1:k}) \times p(Z \rightarrow w_{k+1:j}) \quad [10.12]$$

$$= p(X \rightarrow w_{i+1:j}). \quad [10.13]$$

5185 The cell $t[0, M, S]$ is then the probability of deriving the entire input, $p(S \rightarrow w)$. The name
 5186 “inside recurrence” implies a corresponding **outside recurrence**, which computes the
 5187 probability of a non-terminal X spanning $w_{i+1:j}$, joint with the outside context $(w_{1:i}, w_{j+1:M})$.
 5188 This recurrence is described in § 10.4.3.

5189 10.4 Learning weighted context-free grammars

5190 Like sequence labeling, context-free parsing is a form of structure prediction. As a result,
 5191 WCFGs can be learned using the same set of algorithms: generative probabilistic models,
 5192 structured perceptron, maximum conditional likelihood, and maximum margin learning.
 5193 In all cases, learning requires a **treebank**, which is a dataset of sentences labeled with
 5194 context-free parses. Parsing research was catalyzed by the **Penn Treebank** (Marcus et al.,
 5195 1993), the first large-scale dataset of this type (see § 9.2.2). Phrase structure treebanks exist
 5196 for roughly two dozen other languages, with coverage mainly restricted to European and
 5197 East Asian languages, plus Arabic and Urdu.

5198 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:

$$\psi(X \rightarrow \alpha) = p(X \rightarrow \alpha) \quad [10.14]$$

$$\hat{p}(X \rightarrow \alpha) = \frac{\text{count}(X \rightarrow \alpha)}{\text{count}(X)}. \quad [10.15]$$

5199 For example, the probability of the production $\text{NP} \rightarrow \text{DET NN}$ is the corpus count of
 5200 this production divided by the count of the non-terminal NP . This estimator applies to
 5201 terminal productions as well: the probability of $\text{NN} \rightarrow \text{whale}$ is the count of how often
 5202 *whale* appears in the corpus as generated from an NN tag, divided by the total count of
 5203 the NN tag. Even with the largest treebanks — currently on the order of one million tokens
 5204 — it is difficult to accurately compute probabilities of even moderately rare events, such
 5205 as $\text{NN} \rightarrow \text{whale}$. Therefore, smoothing is critical for making PCFGs effective.

5206 **10.4.2 Feature-based parsing**

5207 The scores for each production can be computed as an inner product of weights and fea-
 5208 tures,

$$\psi(X \rightarrow \alpha) = \boldsymbol{\theta} \cdot \mathbf{f}(X, \alpha), \quad [10.16]$$

5209 where the feature vector $\mathbf{f}(X, \alpha)$ is a function of the left-hand side X and the right-hand
 5210 side α . The basic feature $\mathbf{f}(X, \alpha) = \{(X, \alpha)\}$ encodes only the identity of the production
 5211 itself, which is a discriminatively-trained model with the same expressiveness as a PCFG.
 5212 Other features can be obtained by grouping elements on either the left-hand or right-
 5213 hand side. It can be particularly beneficial to compute additional features by clustering
 5214 terminal symbols, with features corresponding to groups of words with similar syntactic
 5215 properties. The clustering can be obtained from unlabeled datasets that are much larger
 5216 than any treebank, improving coverage. Such methods are described in chapter 14.

It is also possible to build features that include lexical information about the span and its boundaries. Such features are defined over **anchored productions**,

$$\psi(X \rightarrow \alpha, (i, j, k)) = \boldsymbol{\theta} \cdot \mathbf{f}(X, \alpha, (i, j, k), \mathbf{w})$$

5217 where the feature vector is now a function of the details of the production (X and α),
 5218 as well as the text \mathbf{w} and the indices of the spans to derive: the parent $\mathbf{w}_{i+1:j}$, the left
 5219 child $\mathbf{w}_{i+1:k}$, and the right child $\mathbf{w}_{k+1:j}$. Features on anchored productions can include
 5220 the words that border the span w_i, w_{j+1} , the word at the split point w_{k+1} , the presence of
 5221 a verb or noun in the left child span $w_{i+1:k}$, and so on (Durrett and Klein, 2015). Strictly
 5222 speaking, the WCFG formalism scores only the productions $X \rightarrow \alpha$, and not anchored
 5223 productions. However, scores on anchored productions can be incorporated into CKY
 5224 parsing without any modification to the algorithm, because it is still possible to compute
 5225 each element of the table $t[i, j, X]$ recursively from its immediate children.

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),

$$\mathbf{f}(\tau, \mathbf{w}^{(i)}) = \sum_{(X \rightarrow \alpha, (i, j, k)) \in \tau} \mathbf{f}(X, \alpha, (i, j, k), \mathbf{w}^{(i)}) \quad [10.17]$$

$$\hat{\tau} = \operatorname{argmax}_{\tau: \text{yield}(\tau) = \mathbf{w}^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\tau, \mathbf{w}^{(i)}) \quad [10.18]$$

$$\boldsymbol{\theta} \leftarrow \mathbf{f}(\hat{\tau}, \mathbf{w}^{(i)}) - \mathbf{f}(\hat{\tau}, \mathbf{w}^{(i)}). \quad [10.19]$$

5226 A margin-based objective can be optimized by selecting $\hat{\tau}$ through cost-augmented decod-
 5227 ing (§ 2.3.2), enforcing a margin of $\Delta(\hat{\tau}, \tau)$ between the hypothesis and the reference parse,
 5228 where Δ is a non-negative cost function, such as the Hamming loss (Stern et al., 2017). It
 5229 is also possible to train feature-based parsing models by conditional log-likelihood, as
 5230 described next.

5231 **10.4.3 *Conditional random field parsing**

5232 The score of a derivation $\Psi(\tau)$ can be converted into a probability by normalizing over all
 5233 possible derivations,

$$p(\tau \mid \mathbf{w}) = \frac{\exp \Psi(\tau)}{\sum_{\tau': \text{yield}(\tau')=\mathbf{w}} \exp \Psi(\tau')}.$$
[10.20]

5234 This suggests another possible objective for learning a WCFG, maximizing the conditional
 5235 log-likelihood of a labeled corpus.

5236 Just as in logistic regression and the sequence CRF, the gradient of the conditional
 5237 log-likelihood is the difference between the observed and expected counts of each fea-
 5238 ture. The expectation $E_{\tau \mid \mathbf{w}}[\mathbf{f}(\tau, \mathbf{w}^{(i)}); \theta]$ requires summing over all possible parses, and
 5239 computing the marginal probabilities of anchored productions, $p(X \rightarrow \alpha, (i, j, k) \mid \mathbf{w})$.
 5240 In CRF sequence labeling, similar marginal probabilities are computed by the two-pass
 5241 **forward-backward algorithm** (§ 7.5.3.3). The analogue for context-free grammars is the
 5242 **inside-outside algorithm**, in which marginal probabilities are computed from terms gen-
 5243 erated by an upward and downward pass over the parsing chart:

- The upward pass is performed by the **inside recurrence**, which is described in § 10.3.2. It computes the score,

$$\alpha(i, j, X) \triangleq \bigoplus_{(X \rightarrow Y Z)} \bigoplus_{k=i}^j \psi(X \rightarrow Y Z, (i, j, k)) \otimes \alpha(i, k, Y) \otimes \alpha(k, j, Z).$$
[10.21]

5244 The initial condition of this recurrence is $\alpha(i-1, i, X) = \psi(X \rightarrow w_i)$. To compute
 5245 log-probabilities, the recurrence is applied in a semiring where $a \otimes b = a + b$ and
 5246 $a \oplus b = \log(\exp x + \exp y)$. Thus, if $\alpha(i, k, Y)$, $\alpha(k, j, Z)$, and $\Psi(X \rightarrow Y Z)$ are
 5247 log-probabilities, then $\alpha(i, j, X)$ will be a log-probability as well. The denominator
 5248 $\sum_{\tau: \text{yield}(\tau)=\mathbf{w}} \exp \Psi(\tau)$ is equal to $\exp \alpha(0, M, S)$.

- The downward pass is performed by the **outside recurrence**, which recursively pop-
 ulates the same chart structure, starting at the root of the tree. It computes the prob-
 ility,

$$\beta(i, j, X) \triangleq \left(\bigoplus_{(Y \rightarrow Z X)} \bigoplus_{k=0}^i \psi(Y \rightarrow Z X, (k, i, j)) \otimes \alpha(k, i, Z) \otimes \beta(k, j, Y) \right)$$
[10.22]

$$\oplus \left(\bigoplus_{(Y \rightarrow X Z)} \bigoplus_{k=j+1}^M \Psi(Y \rightarrow X Z, (i, k, j)) \otimes \alpha(j+1, k, Z) \otimes \beta(i, k, Y) \right).$$
[10.23]

5249 [todo: check for off-by-one errors] The first line of Equation 10.23 is the score under
 5250 the condition that X is a right child of its parent Y , which spans $w_{k:j}$, with $k < i$;
 5251 the second line is the score under the condition that X is a left child of its parent,
 5252 which spans $w_{i:k}$, with $k > j$. In each case, we sum over all possible productions
 5253 with X on the right-hand side. The parent Y is bounded on one side by either i or j ,
 5254 depending on whether X is a left or right child of Y ; we must sum over all possible
 5255 values for the other boundary.

5256 The initial condition for the outside recurrence is $\beta(1, M, S) = \bar{1}$, where $\bar{1}$ is the mul-
 5257 tiplicative identity, $x \otimes \bar{1} = x$, and $\beta(1, M, X \neq S) = \bar{0}$, where $\bar{0}$ is the multiplicative
 5258 annihilator, $x \otimes \bar{0} = \bar{0}$. In the log-probability semiring, $\bar{1} = 0$, and $\bar{0} = -\infty$.

The marginal probability of a non-terminal X over span $w_{i:j}$ is written $p(X \rightsquigarrow w_{i:j} \mid w)$, and depends on the sum of the scores of all parses τ that derive the span $w_{i:j}$ from the non-terminal X , which is computed as $\alpha(i, j, X) \otimes \beta(i, j, X)$. This can be converted to a probability by exponentiating and normalizing,

$$p(X \rightsquigarrow w_{i:j} \mid w_{1:i-1}, w_{j+1:M}) = \frac{\exp(\alpha(i, j, X) \otimes \beta(i, j, X))}{\exp(\alpha(1, M, S))}. \quad [10.24]$$

5259 Probabilities of individual productions can be computed similarly (see exercises). The
 5260 same marginal probabilities can be used in unsupervised **grammar induction**, in which
 5261 a PCFG is learned from a dataset of unlabeled text (Lari and Young, 1990; Pereira and
 5262 Schabes, 1992).

5263 10.4.4 Neural context-free grammars

5264 Recent work has applied neural representations to parsing, representing each span with
 5265 a dense numerical vectors (Socher et al., 2013; Durrett and Klein, 2015; Cross and Huang,
 5266 2016).³ For example, the anchor (i, j, k) and sentence w can be associated with a fixed-
 5267 length column vector,

$$\mathbf{v}_{(i,j,k)} = [\mathbf{u}_{w_{i-1}}; \mathbf{u}_{w_i}; \mathbf{u}_{w_{j-1}}; \mathbf{u}_{w_j}; \mathbf{u}_{w_{k-1}}; \mathbf{u}_{w_k}], \quad [10.25]$$

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),

$$\tilde{\mathbf{v}}_{(i,j,k)} = \text{FeedForward}(\mathbf{v}_{(i,j,k)}) \quad [10.26]$$

$$\psi(X \rightarrow \alpha, (i, j, k)) = \tilde{\mathbf{v}}_{(i,j,k)}^\top \Theta f(X \rightarrow \alpha), \quad [10.27]$$

³Earlier 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).

5268 where $f(X \rightarrow \alpha)$ is a vector of discrete features of the production, and Θ is a parameter
 5269 matrix. The matrix Θ and the parameters of the feedforward network can be learned by
 5270 backpropagating from an objective such as the margin loss or the negative conditional
 5271 log-likelihood.

5272 10.5 Grammar refinement

5273 The strength of the independence assumptions underlying CFG parsing depends on the
 5274 granularity of the non-terminals. For the Penn Treebank non-terminals, there are several
 5275 reasons to believe that these assumptions are too strong to enable accurate parsing (John-
 5276 son, 1998):

- 5277 • The context-free assumption is too strict: for example, the probability of the produc-
 5278 tion $NP \rightarrow NP\ PP$ is much higher (in the PTB) if the parent of the noun phrase is a
 5279 verb phrase (indicating that the NP is a direct object) than if the parent is a sentence
 5280 (indicating that the NP is the subject of the sentence).
- 5281 • The Penn Treebank non-terminals are too coarse: there are many kinds of noun
 5282 phrases and verb phrases, and accurate parsing sometimes requires knowing the
 5283 difference. As we have already seen, when faced with prepositional phrase at-
 5284 tachment ambiguity, a weighted CFG will either always choose NP attachment (if
 5285 $\psi(NP \rightarrow NP\ PP) > \psi(VP \rightarrow VP\ PP)$), or it will always choose VP attachment. To
 5286 get more nuanced behavior, more fine-grained non-terminals are needed.
- 5287 • More generally, accurate parsing requires some amount of **semantics** — understand-
 5288 ing the meaning of the text to be parsed. Consider the example *cats scratch people with*
 5289 *claws*: knowledge of about *cats*, *claws*, and scratching is necessary to correctly resolve
 5290 the attachment ambiguity.

5291 An extreme example is shown in Figure 10.3. The analysis on the left is preferred
 5292 because of the conjunction of similar entities *France* and *Italy*. But given the non-terminals
 5293 shown in the analyses, there is no way to differentiate these two parses, since they include
 5294 exactly the same productions. What is needed seems to be more precise non-terminals.
 5295 One possibility would be to rethink the linguistics behind the Penn Treebank, and ask
 5296 the annotators to try again. But the original annotation effort took five years, and there
 5297 is a little appetite for another annotation effort of this scope. Researchers have therefore
 5298 turned to automated techniques.

5299 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 neigh-
 bors. The validity of this assumption is an empirical question, and it depends on the

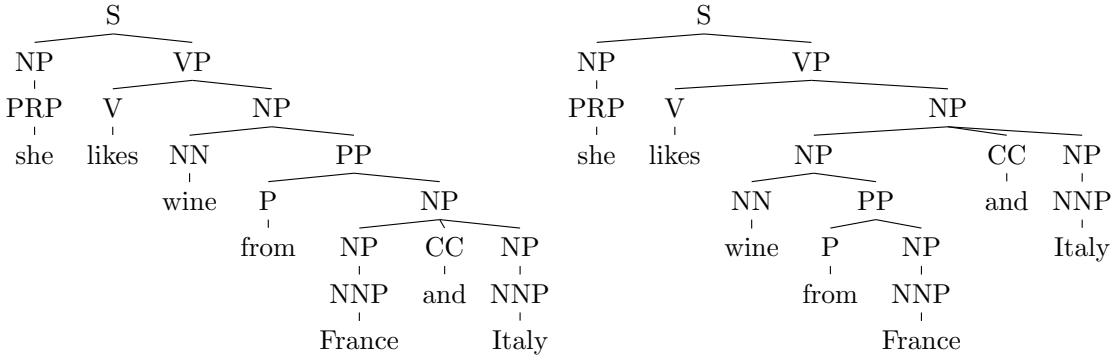


Figure 10.3: The left parse is preferable because of the conjunction of phrases headed by *France* and *Italy*, but these parses cannot be distinguished by a WCFG.

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):

$$P(\text{NP} \rightarrow \text{NP PP}) = 11\% \quad [10.28]$$

$$P(\text{NP UNDER S} \rightarrow \text{NP PP}) = 9\% \quad [10.29]$$

$$P(\text{NP UNDER VP} \rightarrow \text{NP PP}) = 23\% \quad [10.30]$$

5300 This phenomenon can be captured by **parent annotation** (Johnson, 1998), in which each
 5301 non-terminal is augmented with the identity of its parent, as shown in Figure 10.4. This is
 5302 sometimes called **vertical Markovization**, since a Markov dependency is introduced be-
 5303 tween each node and its parent (Klein and Manning, 2003). It is analogous to moving from
 5304 a bigram to a trigram context in a hidden Markov model. In principle, parent annotation
 5305 squares the size of the set of non-terminals, which could make parsing considerably less
 5306 efficient. But in practice, the increase in the number of non-terminals that actually appear
 5307 in the data is relatively modest (Johnson, 1998).

5308 Parent annotation weakens the WCFG independence assumptions. This improves ac-
 5309 curacy by enabling the parser to make more fine-grained distinctions, which better cap-
 5310 ture real linguistic phenomena. However, each production is more rare, and so careful
 5311 smoothing or regularization is required to control the variance over production scores.

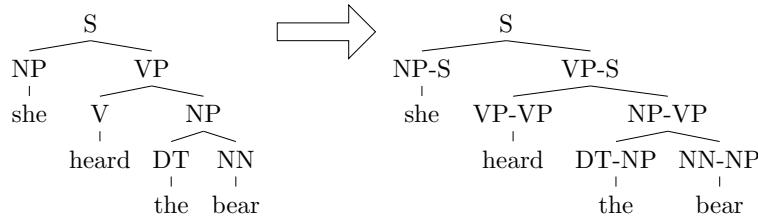


Figure 10.4: Parent annotation in a CFG derivation

5312 10.5.2 Lexicalized context-free grammars

5313 The examples in § 10.2.2 demonstrate the importance of individual words in resolving
 5314 parsing ambiguity: the preposition *on* is more likely to attach to *met*, while the preposi-
 5315 tion *of* is more likely to attach to *President*. But of all word pairs, which are relevant
 5316 to attachment decisions? Consider the following variants on the original examples:

- 5317 (10.4) We met the President of Mexico.
- 5318 (10.5) We met the first female President of Mexico.
- 5319 (10.6) They had supposedly met the President on Monday.

5320 The underlined words are the **head words** of their respective phrases: *met* and *meet* are the
 5321 heads of the verb phrase, and *President* is the head of the direct object noun phrase. These
 5322 heads provide useful semantic information. But they break the context-free assumption,
 5323 which states that the score for a production depends only on the parent and its immediate
 5324 children, and not 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,

$$\text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(\text{of}) \quad [10.31]$$

$$\text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(\text{on}). \quad [10.32]$$

5325 Lexicalization was a major step towards accurate PCFG parsing. It requires solving three
 5326 problems: identifying the heads of all constituents in a treebank; parsing efficiently while
 5327 keeping track of the heads; and estimating the scores for lexicalized productions.

5328 10.5.2.1 Identifying head words

5329 The head of a constituent is the word that is the most useful for determining how that
 5330 constituent is integrated into the rest of the sentence.⁴ The head word of a constituent is

⁴This is a pragmatic definition, befitting our goal of using head words to improve parsing; for a more formal definition, see (Bender, 2013, chapter 7).

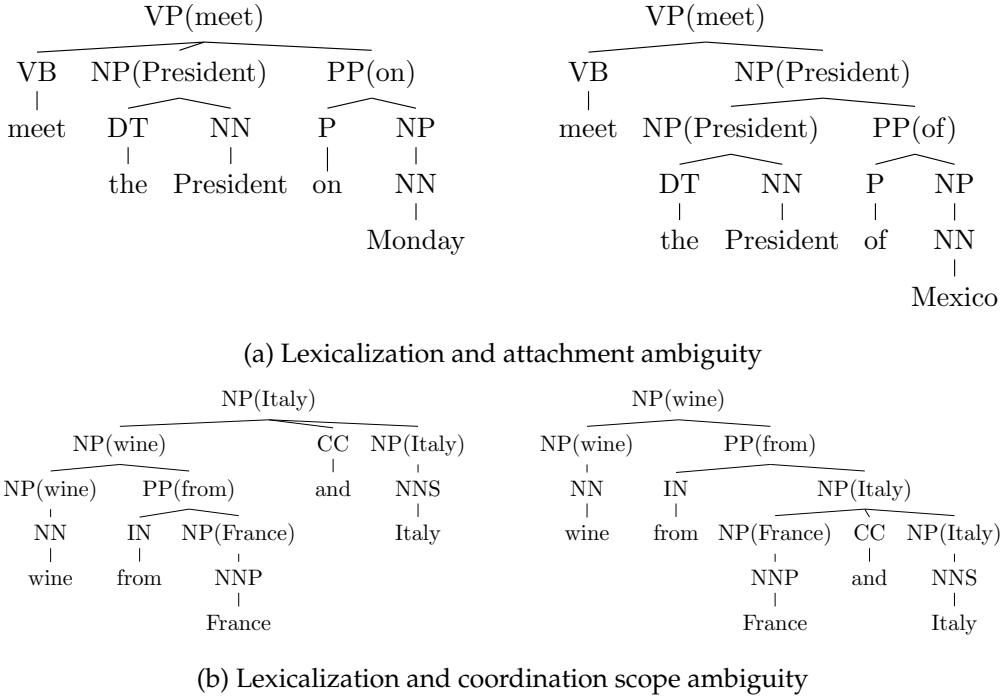


Figure 10.5: Examples of lexicalization

5331 determined recursively: for any non-terminal production, the head of the left-hand side
 5332 must be the head of one of the children. The head is typically selected according to a set of
 5333 deterministic rules, sometimes called **head percolation rules**. In many cases, these rules
 5334 are straightforward: the head of a noun phrase in a $NP \rightarrow DET\ NN$ production is the head
 5335 of the noun; the head of a sentence in a $S \rightarrow NP\ VP$ production is the head of the verb
 5336 phrase.

5337 Table 10.3 shows a fragment of the head percolation rules used in many English pars-
 5338 ing systems. The meaning of the first rule is that to find the head of an S constituent, first
 5339 look for the rightmost VP child; if you don't find one, then look for the rightmost $SBAR$
 5340 child, and so on down the list. Verb phrases are headed by left verbs (the head of *can plan*
 5341 *on walking* is *planned*, since the modal verb *can* is tagged *MD*); noun phrases are headed by
 5342 the rightmost noun-like non-terminal (so the head of *the red cat* is *cat*),⁵ and prepositional
 5343 phrases are headed by the preposition (the head of *at Georgia Tech* is *at*). Some of these
 5344 rules are somewhat arbitrary — there's no particular reason why the head of *cats and dogs*

⁵The 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.

(c) Jacob Eisenstein 2018. Work in progress.

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>

5345 should be *dogs* — but the point here is just to get some lexical information that can support
 5346 parsing, not to make deep claims about syntax. Figure 10.5 shows the application of these
 5347 rules to two of the running examples.

5348 10.5.2.2 Parsing lexicalized context-free grammars

5349 A naïve application of lexicalization would simply increase the set of non-terminals by
 5350 taking the cross-product with the set of terminal symbols, so that the non-terminals now
 5351 include symbols like $\text{NP}(\text{President})$ and $\text{VP}(\text{meet})$. Under this approach, the CKY parsing
 5352 algorithm could be applied directly to the lexicalized production rules. However, the
 5353 complexity would be cubic in the size of the vocabulary of terminal symbols, which would
 5354 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 non-terminal X spanning $w_{i+1:j}$, with head word h , where $i < h \leq j$. To compute such a table recursively, we must consider the possibility that each cell 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:

$$t_\ell[i, j, h, X] = \max_{(X \rightarrow Y Z)} \max_{k > h} \max_{k < h' \leq j} t[i, k, h, Y] \otimes t[k, j, h', Z] \otimes \psi(X(h) \rightarrow Y(h)Z(h')) \quad [10.33]$$

$$t_r[i, j, h, X] = \max_{(X \rightarrow Y Z)} \max_{k < h} \max_{i < h' \leq k} t[i, k, h', Y] \otimes t[k, j, h, Z] \otimes (\psi(X(h) \rightarrow Y(h')Z(h))) \quad [10.34]$$

$$t[i, j, h, X] = \max(t_\ell[i, j, h, X], t_r[i, j, h, X]). \quad [10.35]$$

5355 To compute t_ℓ , we maximize over all split points $k > h$, since the head word must be in
 5356 the left child. We then maximize again over possible head words h' for the right child. An
 5357 analogous computation is performed for t_r . The size of the table is now $\mathcal{O}(M^3 N)$, where
 5358 M is the length of the input and N is the number of non-terminals. Furthermore, each

5359 cell is computed by performing $\mathcal{O}(M^2)$ operations, since we maximize over both the split
 5360 point k and the head h' . The time complexity of the algorithm is therefore $\mathcal{O}(RM^5N)$,
 5361 where R is the number of rules in the grammar. Fortunately, more efficient solutions are
 5362 possible. In general, the complexity of parsing can be reduced to $\mathcal{O}(M^4)$ in the length of
 5363 the input; for a broad class of lexicalized CFGs, the complexity can be made cubic in the
 5364 length of the input, just as in unlexicalized CFGs (Eisner, 2000).

5365 **10.5.2.3 Estimating lexicalized context-free grammars**

5366 The final problem for lexicalized parsing is how to estimate weights for lexicalized pro-
 5367 ductions $X(i) \rightarrow Y(j) Z(k)$. These productions are said to be **bilexical**, because they
 5368 involve scores over pairs of words: in the example *meet the President of Mexico*, we hope
 5369 to choose the correct attachment point by modeling the bilexical affinities of (*meet, of*) and
 5370 (*President, of*). The number of such word pairs is quadratic in the size of the vocabulary,
 5371 making it difficult to estimate the weights of lexicalized production rules directly from
 5372 data. This is especially true for probabilistic context-free grammars, in which the weights
 5373 are obtained from smoothed relative frequency. In a treebank with a million tokens, a
 5374 vanishingly small fraction of the possible lexicalized productions will be observed more
 5375 than once.⁶ The Charniak (1997) and Collins (1997) parsers therefore focus on approxi-
 5376 mating the probabilities of lexicalized productions, using various smoothing techniques
 5377 and independence assumptions.

In discriminatively-trained weighted context-free grammars, the weights for each produc-
 tion 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
 $NP(President) \rightarrow NP(President) PP(of)$ can be computed from the following features:

$$\begin{aligned} f(NP(President) \rightarrow NP(President) PP(of)) = & \{NP(*) \rightarrow NP(*) PP(*), \\ & NP(President) \rightarrow NP(President) PP(*), \\ & NP(*) \rightarrow NP(*) PP(of), \\ & NP(President) \rightarrow NP(President) PP(of)\} \end{aligned}$$

5378 The first feature scores the unlexicalized production $NP \rightarrow NP PP$; the next two features
 5379 lexicalize only one element of the production, thereby scoring the appropriateness of NP
 5380 attachment for the individual words *President* and *of*; the final feature scores the specific
 5381 bilexical affinity of *President* and *of*. For bilexical pairs that are encountered frequently in
 5382 the treebank, this bilexical feature can play an important role in parsing; for pairs that are
 5383 absent or rare, regularization will drive its weight to zero, forcing the parser to rely on the
 5384 more coarse-grained features.

⁶The 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).

5385 In chapter 14, we will encounter techniques for clustering words based on their **distribu-**
 5386 **tional** properties — the contexts in which they appear. Such a clustering would group
 5387 rare and common words, such as *whale*, *shark*, *beluga*, *Leviathan*. Word clusters can be used
 5388 as features in discriminative lexicalized parsing, striking a middle ground between full
 5389 lexicalization and non-terminals (Finkel et al., 2008). In this way, labeled examples con-
 5390 taining relatively common words like *whale* can help to improve parsing for rare words
 5391 like *beluga*, as long as those two words are clustered together.

5392 10.5.3 *Refinement grammars

5393 Lexicalization improves on context-free parsing by adding detailed information in the
 5394 form of lexical heads. However, estimating the scores of lexicalized productions is dif-
 5395 ficult. Klein and Manning (2003) argue that the right level of linguistic detail is some-
 5396 where between treebank categories and individual words. Some parts-of-speech and non-
 5397 terminals are truly substitutable: for example, *cat*/N and *dog*/N. But others are not: for
 5398 example, the preposition *of* exclusively attaches to nouns, while the preposition *as* is more
 5399 likely to modify verb phrases. Klein and Manning (2003) obtained a 2% improvement in
 5400 *F*-MEASURE on a parent-annotated PCFG parser by making a single change: splitting the
 5401 preposition category into six subtypes. They propose a series of linguistically-motivated
 5402 refinements to the Penn Treebank annotations, which in total yielded a 40% error reduc-
 5403 tion.

5404 Non-terminal refinement process can be automated by treating the refined categories
 5405 as latent variables. For example, we might split the noun phrase non-terminal into NP1, NP2, NP3, ...,
 5406 without defining in advance what each refined non-terminal corresponds to. This can
 5407 be treated as **partially supervised learning**, similar to the multi-component document
 5408 classification model described in § 5.2.3. A latent variable PCFG can be estimated by
 5409 expectation-maximization (Matsuzaki et al., 2005):

- In the E-step, estimate a marginal distribution q over the refinement type of each non-terminal in each derivation. These marginals are constrained by the original annotation: an NP can be reannotated as NP4, but not as VP3. Marginal probabilities over refined productions can be computed from the **inside-outside algorithm**, as described in § 10.4.3. In the special case of a PCFG, the inside and outside recur-

rences have the following probabilistic interpretations:

$$\alpha(i, j, X) = \sum_{(X \rightarrow Y Z)} \sum_{k=i+1}^j \psi(X \rightarrow Y Z) \times \alpha(i, k, Y) \times \alpha(k, j, Z) \quad [10.36]$$

$$= p(\mathbf{w}_{i+1:j} | X) \quad [10.37]$$

$$\begin{aligned} \beta(i, j, X) &= \sum_{(Y \rightarrow Z X)} \sum_{k=0}^i \psi(Y \rightarrow Z X) \times \alpha(k, i, Z) \times \beta(k, j, Y) \\ &\quad + \sum_{(Y \rightarrow X Z)} \sum_{k=j+1}^M \psi(Y \rightarrow X Z) \times \alpha(j, k, Z) \times \beta(i, k, Y) \end{aligned} \quad [10.38]$$

$$= p(\mathbf{w}_{1:i}, \mathbf{w}_{j+1:M}, X). \quad [10.39]$$

From the inside and outside variables, it is possible to compute the marginal probabilities of anchored productions:

$$p(X \rightarrow Y Z, (i, j, k) | \mathbf{w}) = \frac{\beta(i, j, X) \times \psi(X \rightarrow Y Z) \times \alpha(i, k, Y) \times \alpha(k, j, Z)}{\alpha(0, M, S)}. \quad [10.40]$$

- 5410 • In the M-step, recompute the parameters of the grammar, by summing over the
5411 probabilities of anchored productions,

$$E[\text{count}(X \rightarrow Y Z)] = \sum_{i,j,k} p(X \rightarrow Y Z, (i, j, k) | \mathbf{w}). \quad [10.41]$$

5412 As usual, this process can be iterated to convergence. To determine the number of re-
5413 finement types for each tag, Petrov et al. (2006) apply a split-merge heuristic; Liang et al.
5414 (2007) and Finkel et al. (2007) apply **Bayesian nonparametrics** (Cohen, 2016).

5415 Some examples of refined non-terminals are shown in Table 10.4. The proper nouns
5416 differentiate months, first names, middle initials, last names, first names of places, and
5417 second names of places; each of these will tend to appear in different parts of grammatical
5418 productions. The personal pronouns differentiate grammatical role, with PRP-0 appear-
5419 ing in subject position at the beginning of the sentence (note the capitalization), PRP-1
5420 appearing in subject position but not at the beginning of the sentence, and PRP-2 appear-
5421 ing in object position.

5422 10.6 Beyond context-free parsing

5423 In the context-free setting, the score for a parse is a combination of the scores of individual
5424 productions. As we have seen, these models can be improved by using finer-grained non-
5425 terminals, via parent-annotation, lexicalization, and automated refinement. However, the

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).

5426 inherent limitations to the expressiveness of context-free parsing motivate the consider-
 5427 ation of other search strategies. These strategies abandon the optimality guaranteed by
 5428 bottom-up parsing, in exchange for the freedom to consider arbitrary properties of the
 5429 proposed parses.

5430 **10.6.1 Reranking**

5431 A simple way to relax the restrictions of context-free parsing is to perform a two-stage pro-
 5432 cess, in which a context-free parser generates a k -best list of candidates, and a **reranker**
 5433 then selects the best parse from this list (Charniak and Johnson, 2005; Collins and Koo,
 5434 2005). The reranker can be trained from an objective that is similar to multi-class classi-
 5435 fication: the goal is to learn weights that assign a high score to the reference parse, or to
 5436 the parse on the k -best list that has the lowest error. In either case, the reranker need only
 5437 evaluate the K best parses, and so no context-free assumptions are necessary. This opens
 5438 the door to more expressive scoring functions:

- 5439 • It is possible to incorporate arbitrary non-local features, such as the structural par-
 5440 allelism and right-branching orientation of the parse (Charniak and Johnson, 2005).
- 5441 • Reranking enables the use of **recursive neural networks**, in which each constituent
 5442 span $w_{i:j}$ receives a vector $u_{i:j}$ which is computed from the vector representations of
 5443 its children, using a composition function that is linked to the production rule (Socher
 5444 et al., 2013), e.g.,

$$u_{i:j} = f \left(\Theta_{X \rightarrow Y} Z \begin{bmatrix} u_{i:k-1} \\ u_{k:j} \end{bmatrix} \right) \quad [10.42]$$

(c) Jacob Eisenstein 2018. Work in progress.

5445 The overall score of the parse can then be computed from the final vector, $\Psi(\tau) =$
 5446 $\theta \mathbf{u}_{1:M}$.

5447 Reranking can yield substantial improvements in accuracy. The main limitation is that it
 5448 can only find the best parse among the K -best offered by the generator, so it is inherently
 5449 limited by the ability of the bottom-up parser to find high-quality candidates.

5450 **10.6.2 Transition-based parsing**

5451 Structure prediction can be viewed as a form of search. An alternative search strategy is to
 5452 read the input from left-to-right, gradually building up a parse structure through a series
 5453 of **transitions**. Transition-based parsing is described in more detail in the next chapter, in
 5454 the context of dependency parsing. However, it can also be applied to CFG parsing, as
 5455 briefly described here.

5456 For any context-free grammar, there is an equivalent **pushdown automaton**, a model
 5457 of computation that accepts exactly those strings that can be derived from the grammar.
 5458 This computational model consumes the input from left to right, while pushing and pop-
 5459 ping elements on a stack. This architecture provides a natural transition-based parsing
 5460 framework for context-free grammars, known as **shift-reduce parsing**.

5461 Shift-reduce parsing is a type of transition-based parsing, in which the parser can take
 5462 the following actions:

- 5463 • *shift* the next terminal symbol onto the stack;
- 5464 • *unary-reduce* the top item on the stack, using a unary production rule in the gram-
 mar;
- 5466 • *binary-reduce* the top two items onto the stack, using a binary production rule in the
 grammar.

5468 The set of available actions is constrained by the situation: the parser can only shift if
 5469 there are remaining terminal symbols in the input, and it can only reduce if an applicable
 5470 production rule exists in the grammar. If the parser arrives at a state where the input
 5471 has been completely consumed, and the stack contains only the element S, then the input
 5472 is accepted. If the parser arrives at a non-accepting state where there are no possible
 5473 actions, the input is rejected. A parse error occurs if there is some action sequence that
 5474 would accept an input, but the parser does not find it.

5475 **Example** Consider the input *we eat sushi* and the grammar in Table 10.1. The input can
 5476 be parsed through the following sequence of actions:

- 5477 1. **Shift** the first token *we* onto the stack.

- 5478 2. **Reduce** the top item on the stack to NP, using the production $\text{NP} \rightarrow we$.
 5479 3. **Shift** the next token *eat* onto the stack, and **reduce** it to V with the production $\text{V} \rightarrow$
 5480 *eat*.
 5481 4. **Shift** the final token *sushi* onto the stack, and **reduce** it to NP. The input has been
 5482 completely consumed, and the stack contains [NP, V, NP].
 5483 5. **Reduce** the top two items using the production $\text{VP} \rightarrow \text{V NP}$. The stack now con-
 5484 tains [VP, NP].
 5485 6. **Reduce** the top two items using the production $\text{S} \rightarrow \text{NP VP}$. The stack now contains
 5486 [S]. Since the input is empty, this is an accepting state.

5487 One thing to notice from this example is that the number of shift actions is equal to the
 5488 length of the input. The number of reduce actions is equal to the number of non-terminals
 5489 in the analysis, which grows linearly in the length of the input. Thus, the overall time
 5490 complexity of shift-reduce parsing is linear in the length of the input (assuming the com-
 5491 plexity of each individual classification decision is constant in the length of the input).
 5492 This is far better than the cubic time complexity required by CKY parsing.

5493 **Transition-based parsing as inference** In general, it is not possible to guarantee that
 5494 a transition-based parser will find the optimal parse, $\text{argmax}_\tau \Psi(\tau; \mathbf{w})$, even under the
 5495 usual CFG independence assumptions. We could assign a score to each anchored parsing
 5496 action in each context, with $\psi(a, c)$ indicating the score of performing action a in context c .
 5497 One might imagine that transition-based parsing could efficiently find the derivation that
 5498 maximizes the sum of such scores. But this too would require backtracking and searching
 5499 over an exponentially large number of possible action sequences: if a bad decision is
 5500 made at the beginning of the derivation, then it may be impossible to recover the optimal
 5501 action sequence without backtracking to that early mistake. This is known as a **search**
 5502 **error**. Transition-based parser can incorporate arbitrary features, without the restrictive
 5503 independence assumptions required by chart parsing; search errors are the price that must
 5504 be paid for this flexibility.

5505 **Learning transition-based parsing** Transition-based parsing can be combined with ma-
 5506 chine learning by training a classifier to select the correct action at each position. This
 5507 classifier is free to choose any feature of the input, the state of the parser, and the parse
 5508 history. However, there is no optimality guarantee: the parser may choose a suboptimal
 5509 parse, due to a mistake at the beginning of the analysis. Nonetheless, some of the strongest
 5510 CFG parsers are based on the shift-reduce architecture, rather than CKY. A recent gener-
 5511 ation of models links shift-reduce parsing with recurrent neural networks, updating a
 5512 hidden state vector while consuming the input (e.g., Cross and Huang, 2016; Dyer et al.,
 5513 2016). Learning algorithms for transition-based parsing are discussed in more detail in
 5514 § 11.3.

5515 **Exercises**

1. Consider the following PCFG:

$$p(X \rightarrow X X) = \frac{1}{2} \quad [10.43]$$

$$p(X \rightarrow Y) = \frac{1}{2} \quad [10.44]$$

$$p(Y \rightarrow \sigma) = \frac{1}{|\Sigma|}, \forall \sigma \in \Sigma \quad [10.45]$$

- 5516 a) Compute the probability $p(\hat{\tau})$ of the maximum probability parse for a string
 5517 $w \in \Sigma^M$.
- 5518 b) Compute the marginal probability $p(w) = \sum_{\tau: \text{yield}(\tau)=w} p(\tau)$.
- 5519 c) Compute the conditional probability $p(\hat{\tau} | w)$.
- 5520 2. Use the inside and outside scores to compute the marginal probability $p(X_{i:j} \rightarrow Y_{i:k-1} Z_{k:j} | w)$,
 5521 indicating that Y spans $w_{i:k-1}$, Z spans $w_{k:j}$, and X is the parent of Y and Z , span-
 5522 ning $w_{i:j}$.
- 5523 3. Suppose that the potentials $\Psi(X \rightarrow \alpha)$ are log-probabilities, so that $\sum_{\alpha} \exp \Psi(X \rightarrow \alpha) = 1$
 5524 for all X . Verify that the semiring inside recurrence from Equation 10.21 generates
 5525 the log-probability $\log p(w) = \log \sum_{\tau: \text{yield}(\tau)=w} p(\tau)$.
- 5526 4. more exercises tk

5527 **Chapter 11**

5528 **Dependency Parsing**

5529 The previous chapter discussed algorithms for analyzing sentences in terms of nested con-
5530 stituents, such as noun phrases and verb phrases. However, many of the key sources of
5531 ambiguity in phrase-structure analysis relate to questions of **attachment**: where to attach
5532 a prepositional phrase or complement clause, how to scope a coordinating conjunction,
5533 and so on. These attachment decisions can be represented with a more lightweight struc-
5534 ture: a directed graph over the words in the sentence, known as a **dependency parse**.
5535 In recent years, syntactic annotation has shifted its focus to such dependency structures:
5536 at the time of this writing, the **Universal Dependencies** project offers more than 100 de-
5537 pendency treebanks for more than 60 languages.¹ This chapter will describe the linguis-
5538 tic ideas underlying dependency grammar, and then discuss exact and transition-based
5539 parsing algorithms. The chapter will also discuss recent research on **learning to search** in
5540 transition-based structure prediction.

5541 **11.1 Dependency grammar**

5542 While **dependency grammar** has a rich history of its own (Tesnière, 1966; Kübler et al.,
5543 2009), it can be motivated by extension from the lexicalized context-free grammars that
5544 we encountered in previous chapter (§ 10.5.2). Recall that lexicalization augments each
5545 non-terminal with a **head word**. The head of a constituent is identified recursively, using
5546 a set of **head rules**, as shown in Table 10.3. An example of a lexicalized context-free parse
5547 is shown in Figure 11.1a. In this sentence, the head of the S constituent is the main verb,
5548 *scratch*; this non-terminal then produces the noun phrase *the cats*, whose head word is
5549 *cats*, and from which we finally derive the word *the*. Thus, the word *scratch* occupies the
5550 central position for the sentence, with the word *cats* playing a supporting role. In turn, *cats*

¹universaldependencies.org

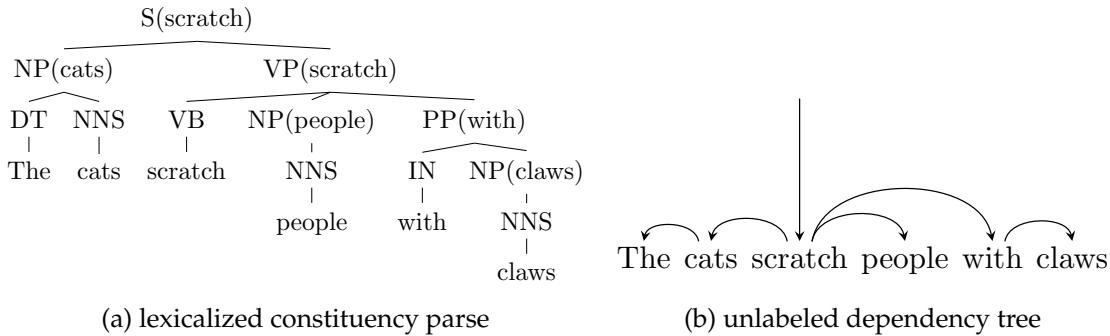


Figure 11.1: Dependency grammar is closely linked to lexicalized context free grammars: each lexical head has a dependency path to every other word in the constituent.

occupies the central position for the noun phrase, with the word *the* playing a supporting role.

These relationships, which hold between the words in the sentence, can be formalized in a directed graph, by placing an edge from word i to word j iff word i is the head of the first branching node above a node headed by j . Thus, in our example, we would have $\text{scratch} \rightarrow \text{cats}$ and $\text{cats} \rightarrow \text{the}$. We would not have the edge $\text{scratch} \rightarrow \text{the}$, because although $S(\text{scratch})$ dominates $\text{DET}(\text{the})$ in the graph, it is not the *first* branching node above the determiner. These edges describe **syntactic dependencies**, a bilexical relationship between a **head** and a **dependent**, which is at the heart of dependency grammar.

If we continue to build out this **dependency graph**, we will eventually reach every word in the sentence, as shown in Figure 11.1b. In this graph — and in all graphs constructed in this way — every word has exactly one incoming edge, except for the root word, which is indicated by a special incoming arrow from above. Furthermore, the graph is *weakly connected*: if the directed edges were replaced with undirected edges, there would be a path between all pairs of nodes. From these properties, it can be shown that there are no cycles in the graph (or else at least one node would have to have more than one incoming edge), and therefore, the graph is a tree. Because the graph includes all vertices, it is a **spanning tree**.

5569 11.1.1 Heads and dependents

5570 A dependency edge implies an asymmetric syntactic relationship between the head and
5571 dependent words, sometimes called **modifiers**. For a pair like *the cats* or *cats scratch*, how
5572 do we decide which is the head? Here are some possible criteria:

- 5573 • The head sets the syntactic category of the construction: for example, nouns are the
5574 heads of noun phrases, and verbs are the heads of verb phrases.

(c) Jacob Eisenstein 2018. Work in progress.

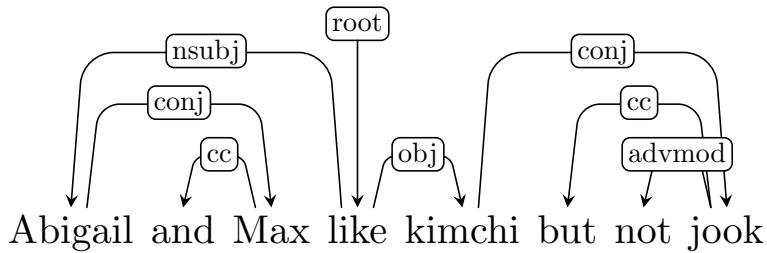


Figure 11.2: In the Universal Dependencies annotation system, the left-most item of a coordination is the head.

- The modifier may be optional while the head is mandatory: for example, in the sentence *cats scratch people with claws*, the subtrees *cats scratch* and *cats scratch people* are grammatical sentences, but *with claws* is not.
- The head determines the morphological form of the modifier: for example, in languages that require gender agreement, the gender of the noun determines the gender of the adjectives and determiners.
- Edges should first connect content words, and then connect function words.

As always, these guidelines sometimes conflict. The Universal Dependencies (UD) project has attempted to identify a set of principles that can be applied to dozens of different languages (Nivre et al., 2016).² These guidelines are based on the universal part-of-speech tags from chapter 8. They differ somewhat from the head rules described in § 10.5.2: for example, on the principle that dependencies should relate content words, the prepositional phrase *with claws* would be headed by *claws*, resulting in an edge *scratch* → *claws*, and another edge *claws* → *with*.

One objection to dependency grammar is that not all syntactic relations are asymmetric. Coordination is one of the most obvious examples (Popel et al., 2013): in the sentence, *Abigail and Max like kimchi* (Figure 11.2), which word is the head of the coordinated noun phrase *Abigail and Max*? Choosing either *Abigail* or *Max* seems arbitrary; fairness argues for the choice of *and*, but this seems the least important word in the noun phrase, and selecting it would violate the principle of linking content words first. The Universal Dependencies annotation system arbitrarily chooses the left-most item as the head — in this case, *Abigail* — and includes edges from this head to both *Max* and the coordinating conjunction *and*. These edges are distinguished by the labels CONJ (for the thing begin conjoined) and CC (for the coordinating conjunction). The labeling system is discussed next.

²The latest and most specific guidelines are available at universaldependencies.org/guidelines.html

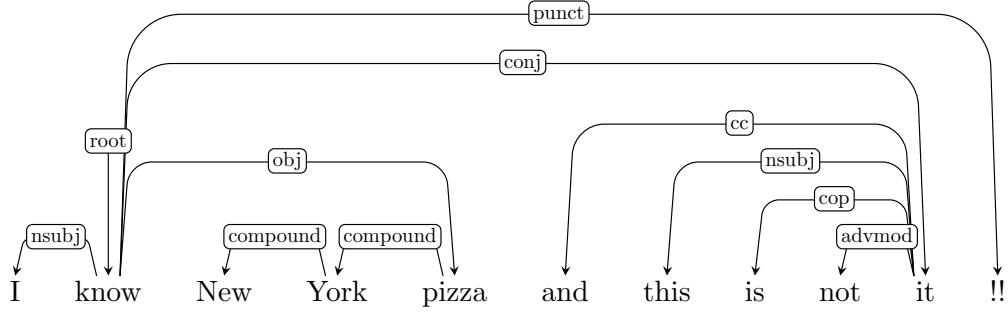


Figure 11.3: A labeled dependency parse from the UD Treebank (reviews-361348-0006)

5600 11.1.2 Labeled dependencies

5601 Edges may be **labeled** to indicate the nature of the syntactic relation that holds between
 5602 the two elements. For example, in Figure 11.2, the label NSUBJ on the edge from *like* to
 5603 *Abigail* indicates that the subtree headed by *Abigail* is the noun subject of the predicate
 5604 verb *like*; similarly, the label OBJ on the edge from *like* to *kimchi* indicates that the sub-
 5605 tree headed by *kimchi* is the object.³ The negation *not* is treated as an adverbial modifier
 5606 (*ADVMOD*) on the noun *jook*.

5607 A slightly more complex example is shown in Figure 11.3. The multiword expression
 5608 *New York Pizza* is treated as a “flat” unit of text, with the elements linked by the COM-
 5609 POUND relation. The sentence includes two clauses that are conjoined in the same way
 5610 that noun phrases are conjoined in Figure 11.2. The second clause contains a **copula** verb
 5611 (see § 8.1.1). For such clauses, we treat the “object” of the verb as the root — in this case,
 5612 *it* — and label the verb as a dependent, with the COP relation. This example also shows
 5613 how punctuations are treated, with label PUNCT.

5614 11.1.3 Dependency subtrees and constituents

5615 Dependency trees hide information that would be present in a CFG parse. Often what
 5616 is hidden is in fact irrelevant: for example, Figure 11.4 shows three different ways of
 5617 representing prepositional phrase adjuncts to the verb *ate*. Because there is apparently
 5618 no meaningful difference between these analyses, the Penn Treebank decides by conven-
 5619 tion to use the two-level representation (see Johnson, 1998, for a discussion). As shown
 5620 in Figure 11.4d, these three cases all look the same in a dependency parse, which is an
 5621 advantage of the dependency representation.

5622 But dependency grammar imposes its own set of annotation decisions, such as the

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.

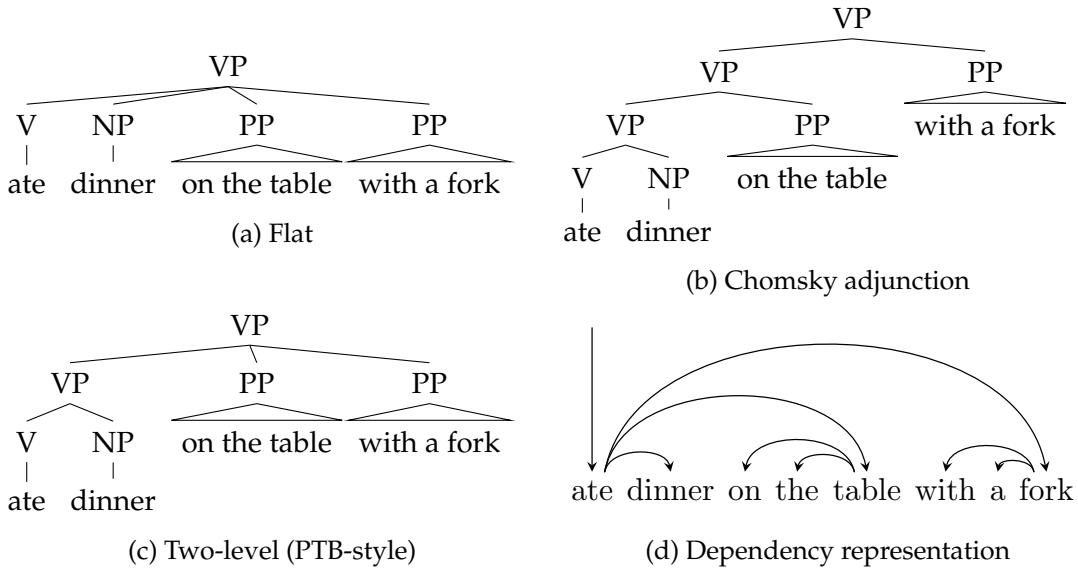


Figure 11.4: The three different CFG analyses of this verb phrase all correspond to a single dependency structure.

5623 identification of the head of a coordination (§ 11.1.1); (unlexicalized) context-free gram-
 5624 mar does not require either element in a coordination to be privileged in this way. Depen-
 5625 dency parses can be disappointingly flat: for example, in the sentence *Yesterday, Abigail*
 5626 *was reluctantly giving Max kimchi*, the root *giving* is the head of every dependency! The
 5627 constituent parse arguably offers a more useful structural analysis for such cases.

5628 **Projectivity** Thus far, we have defined dependency trees as spanning trees over a graph
 5629 in which each word is a vertex. As we have seen, one way to construct such trees is by
 5630 connecting the heads in a lexicalized constituent parse. However, there are spanning trees
 5631 that cannot be constructed in this way. Syntactic constituents are *contiguous spans*. In a
 5632 spanning tree constructed from a lexicalized constituent parse, the head h of any con-
 5633 stituent that spans the nodes from i to j must have a path to every node in this span. This
 5634 is property is known as **projectivity**, and projective dependency parses are a restricted
 5635 class of spanning trees. Informally, projectivity means that “crossing edges” are pro-
 5636 hibited. The formal definition follows:

5637 **Definition 2 (Projectivity).** *An edge from i to j is projective iff all k between i and j are des-
 5638 cendants of i . A dependency parse is projective iff all its edges are projective.*

5639 Figure 11.5 gives an example of a non-projective dependency graph in English. This
 5640 dependency graph does not correspond to any constituent parse. In languages where

	% 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)

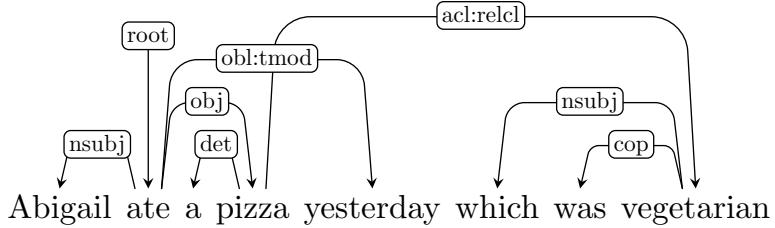


Figure 11.5: An example of a non-projective dependency parse. The “crossing edge” arises from the relative clause *which was vegetarian* and the oblique temporal modifier *yesterday*.

5641 non-projectivity is common, such as Czech and German, it is better to annotate depen-
 5642 dency trees directly, rather than deriving them from constituent parses. An example is the
 5643 Prague Dependency Treebank (Böhmová et al., 2003), which contains 1.5 million words of
 5644 Czech, with approximately 12,000 non-projective edges (see Table 11.1). Even though rel-
 5645 atively few dependencies are non-projective in Czech and German, many sentences have
 5646 at least one such dependency. As we will soon see, projectivity has important algorithmic
 5647 consequences.

5648 11.2 Graph-based dependency parsing

5649 Let $\mathbf{y} = \{(i \rightarrow j, r)\}$ indicate a dependency graph with relation r from head word
 5650 $i \in \{1, 2, \dots, M, \text{ROOT}\}$ to modifier $j \in \{1, 2, \dots, M\}$, where ROOT is a special node indi-
 5651 cating the root of the graph, and M is the length of the input $|\mathbf{w}|$. Given a scoring function
 5652 $\Psi(\mathbf{y}, \mathbf{w}; \theta)$, the optimal parse is,

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}(\mathbf{w})}{\operatorname{argmax}} \Psi(\mathbf{y}, \mathbf{w}; \theta), \quad [11.1]$$

5653 where $\mathcal{Y}(\mathbf{w})$ is the set of valid dependency parses on the input \mathbf{w} . The learning problem
 5654 is to minimize θ with respect to a loss on a labeled dataset, $(\mathbf{y}^{(1:N)}, \mathbf{w}^{(1:N)})$. As usual, the
 5655 number of possible labels $|\mathcal{Y}(\mathbf{w})|$ is exponential in the length of the input (Wu and Chao,
 5656 2004). Algorithms that search over this space of possible graphs are known as **graph-**
 5657 **based dependency parsers**.

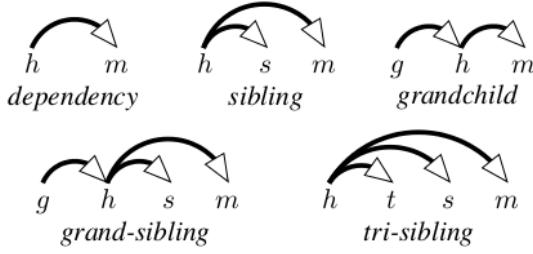


Figure 11.6: Feature templates for higher-order dependency parsing (Koo and Collins, 2010) [todo: permission]

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 $i \rightarrow j$:

$$\Psi(\mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) = \sum_{(i \rightarrow j, r) \in \mathbf{y}} \psi(\mathbf{w}, i \rightarrow j, r; \boldsymbol{\theta}). \quad [11.2]$$

5658 Dependency parsers that operate under this assumption are known as **arc-factored**, since
5659 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,

$$\begin{aligned} \Psi(\mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) = & \sum_{(i \rightarrow j, r) \in \mathbf{y}} \sum_{(k \rightarrow i, r') \in \mathbf{y}} \psi_{\text{grandparent}}(\mathbf{w}, i \rightarrow j, r, (k, r'); \boldsymbol{\theta}) \\ & + \sum_{\substack{(i \rightarrow s, r') \in \mathbf{y} \\ s \neq j}} \psi_{\text{sibling}}(\mathbf{w}, i \rightarrow j, r, (s, r'); \boldsymbol{\theta}). \end{aligned} \quad [11.3]$$

5660 The top line scores computes a scoring function that includes the grandparent k ; the
5661 bottom line computes a scoring function for each sibling s . For projective dependency
5662 graphs, there are efficient algorithms for second-order and third-order dependency pars-
5663 ing (Eisner, 1996; McDonald and Pereira, 2006; Koo and Collins, 2010); for non-projective
5664 dependency graphs, second-order dependency parsing is NP-hard (McDonald and Pereira,
5665 2006). The specific algorithms are discussed in the next section.

5666 **11.2.1 Graph-based parsing algorithms**

5667 The distinction between projective and non-projective dependency trees (§ 11.1.3) plays
 5668 a key role in the choice of algorithms. Because projective dependency trees are closely
 5669 related to (and can be derived from) lexicalized constituent trees, lexicalized parsing al-
 5670 gorithms can be applied directly. For the more general problem of parsing to arbitrary
 5671 spanning trees, a different class of algorithms is required. In both cases, arc-factored de-
 5672 pendency parsing relies on precomputing the scores $\psi(i \rightarrow j, r)$ for each potential edge
 5673 $i \rightarrow j$ with label r . There are $\mathcal{O}(M^2 R)$ such scores, where M is the length of the input and
 5674 R is the number of dependency relation types, and this is a lower bound on the time and
 5675 space complexity of any exact algorithm for arc-factored dependency parsing.

5676 **11.2.1.1 Projective dependency parsing**

5677 As discussed in § 11.1, any lexicalized constituency tree can be converted into a projec-
 5678 tive dependency tree by creating arcs between the heads of constituents and their parents.
 5679 As a result, any algorithm for lexicalized constituent parsing can be converted into an
 5680 algorithm for projective dependency parsing, by converting arc scores into scores for lex-
 5681 icalized productions. As noted in § 10.5.2, there are bottom-up cubic time algorithms for
 5682 lexicalized constituent parsing, which are extensions of the CKY algorithm. Therefore,
 5683 arc-factored projective dependency parsing can be performed in cubic time in the length
 5684 of the input.

5685 Second-order projective dependency parsing can also be performed in cubic time, with
 5686 minimal modifications to the lexicalized parsing algorithm (Eisner, 1996). It is possible to
 5687 go even further, to **third-order dependency parsing**, in which the scoring function may
 5688 consider great-grandparents, grand-siblings, and “tri-siblings”, as shown in Figure 11.6.
 5689 Third-order dependency parsing can be performed in $\mathcal{O}(M^4)$ time, which can be made
 5690 practical through the use of pruning to eliminate unlikely edges (Koo and Collins, 2010).

5691 **11.2.1.2 Non-projective dependency parsing**

5692 In non-projective dependency parsing, the goal is to identify the highest-scoring spanning
 5693 tree over the words in the sentence. The arc-factored assumption ensures that the score for
 5694 each spanning tree will be computed as a sum over scores for the edges, $\psi(i \rightarrow j, r)$, which
 5695 are precomputed. Based on these scores, we build a weighted connected graph. Arc-
 5696 factored non-projective dependency parsing is then equivalent to finding the the spanning
 5697 tree that achieves the maximum total score, $\Psi(\mathbf{y}; \mathbf{w}) = \sum_{(i \rightarrow j, r) \in \mathbf{y}} \psi(i \rightarrow j, r)$. The **Chu-**
Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967) computes this **maximum**
spanning tree efficiently. It does this by first identifying the best incoming edge $i \rightarrow j$ for
 5700 each vertex j . If the resulting graph does not contain cycles, it is the maximum spanning
 5701 tree. If there is a cycle, it is collapsed into a super-vertex, whose incoming and outgoing
 5702 edges are based on the edges to the vertices in the cycle. The algorithm is then applied

5703 recursively to the resulting graph, and process repeats until a graph without cycles is
 5704 obtained.

5705 The time complexity of identifying the best incoming edge for each vertex is $\mathcal{O}(M^2R)$,
 5706 where M is the length of the input and R is the number of relations; in the worst case, the
 5707 number of cycles is $\mathcal{O}(M)$. Therefore, the complexity of the Chu-Liu-Edmonds algorithm
 5708 is $\mathcal{O}(M^3R)$. This complexity can be reduced to $\mathcal{O}(M^2N)$ by storing the edge scores in a
 5709 Fibonacci heap (Gabow et al., 1986). For more detail on graph-based parsing algorithms,
 5710 see Eisner (1997).

5711 **Higher-order non-projective dependency parsing** Given the tractability of higher-order
 5712 projective dependency parsing, you may be surprised to learn that non-projective second-
 5713 order dependency parsing is NP-Hard. This can be proved by reduction from the vertex
 5714 cover problem (Neuhaus and Bröker, 1997). A heuristic solution is to do projective pars-
 5715 ing first, and then post-process the projective dependency parse to add non-projective
 5716 edges (Nivre and Nilsson, 2005). More recent work has applied techniques for approxi-
 5717 mate inference in graphical models, including belief propagation (Smith and Eisner, 2008),
 5718 integer linear programming (Martins et al., 2009), variational inference (Martins et al.,
 5719 2010), and Markov Chain Monte Carlo (Zhang et al., 2014).

5720 11.2.2 Computing scores for dependency arcs

The arc-factored scoring function $\psi(\mathbf{w}, i \rightarrow j, r)$ can be defined in several ways:

$$\text{Linear} \quad \psi(\mathbf{w}, i \rightarrow j, r; \boldsymbol{\theta}) = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, i \rightarrow j, r) \quad [11.4]$$

$$\text{Neural} \quad \psi(\mathbf{w}, i \rightarrow j, r; \boldsymbol{\theta}) = \text{Feedforward}([\mathbf{u}_{w_i}; \mathbf{u}_{w_j}; \mathbf{f}(i, j)]; \boldsymbol{\theta}) \quad [11.5]$$

$$\text{Generative} \quad \psi(\mathbf{w}, i \rightarrow j, r; \boldsymbol{\theta}) = \log p(w_j, r | w_i). \quad [11.6]$$

5721 11.2.2.1 Linear feature-based arc scores

5722 Linear models for dependency parsing incorporate many of the same features used in
 5723 sequence labeling and discriminative constituent parsing. These include:

- 5724 • the length and direction of the arc;
- 5725 • the words w_i and w_j linked by the dependency relation;
- 5726 • the prefixes, suffixes, and part-of-speech of these words;
- 5727 • the neighbors of the dependency arc, $w_{i-1}, w_{i+1}, w_{j-1}, w_{j+1}$;
- 5728 • the prefixes, suffixes, and part-of-speech of these neighbor words.

5729 Each of these features can be conjoined with the dependency edge label r . Note that
 5730 features in an arc-factored parser can refer to words other than w_i and w_j . The restriction
 5731 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:

$$\begin{aligned} f(\text{we eat sushi with chopsticks}, 3, 5) = & \langle \text{sushi} \rightarrow \text{chopsticks}, \\ & \text{sushi} \rightarrow \text{NNS}, \\ & \text{NN} \rightarrow \text{chopsticks}, \\ & \text{NNS} \rightarrow \text{NN} \rangle. \end{aligned}$$

5732 Regularized discriminative learning algorithms can then learn to trade off between fea-
 5733 tures at varying levels of detail. McDonald et al. (2005) take this approach as far as
 5734 tetralexical features (e.g., $(w_i, w_{i+1}, w_{j-1}, w_j)$). Such features help to avoid choosing arcs
 5735 that are unlikely due to the intervening words: for example, there is unlikely to be an
 5736 edge between two nouns if the intervening span contains a verb. A large list of first and
 5737 second-order features is provided by Bohnet (2010), who uses a hashing function to store
 5738 these features efficiently.

5739 11.2.2.2 Neural arc scores

Given vector representations v_i for each word w_i in the input, a set of arc scores can be computed from a feedforward neural network:

$$\psi(i \rightarrow j, r) = \text{FeedForward}([\mathbf{u}_i; \mathbf{u}_j]; \theta_r), \quad [11.7]$$

where unique weights θ_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,

$$\mathbf{z} = g(\Theta_r[\mathbf{u}_i; \mathbf{u}_j] + b_r^{(z)}) \quad [11.8]$$

$$\psi(i \rightarrow j, r) = \beta_r \mathbf{z} + b_r^{(y)}, \quad [11.9]$$

5740 where Θ_r is a matrix, β_r is a vector, each b_r is a scalar, and the function g is an elementwise
 5741 tanh activation function.

The vector \mathbf{u}_i can be set equal to the word embedding v_{w_i} , which may be pre-trained or learned by backpropagation (Pei et al., 2015). Alternatively, contextual information can

be incorporated by applying a bidirectional recurrent neural network across the input,

$$\mathbf{u}_i = [\overrightarrow{\mathbf{u}}_i; \overleftarrow{\mathbf{u}}_i] \quad [11.10]$$

$$\overrightarrow{\mathbf{u}}_i = \overrightarrow{\text{RNN}}(\mathbf{v})_i \quad [11.11]$$

$$\overleftarrow{\mathbf{u}}_i = \overleftarrow{\text{RNN}}(\mathbf{v})_i, \quad [11.12]$$

5742 with \mathbf{v}_i equal to the embedding of word w_i (Kiperwasser and Goldberg, 2016).

5743 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*,

$$\mathbf{y} = \{(ROOT, 2), (2, 1), (2, 3), (3, 5), (5, 4)\} \quad [11.13]$$

$$\log p(\mathbf{w} \mid \mathbf{y}) = \sum_{(i \rightarrow j) \in \mathbf{y}} \log p(w_j \mid w_i) \quad [11.14]$$

$$\begin{aligned} &= \log p(eat \mid ROOT) + \log p(we \mid eat) + \log p(sushi \mid eat) \\ &\quad + \log p(rice \mid sushi) + \log p(with \mid rice). \end{aligned} \quad [11.15]$$

5744 Probabilistic generative models are used in combination with expectation-maximization
5745 (chapter 5) for unsupervised dependency parsing (Klein and Manning, 2004).

5746 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,

$$\hat{\mathbf{y}} = \underset{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})}{\operatorname{argmax}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \mathbf{y}') \quad [11.16]$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \mathbf{f}(\mathbf{w}, \mathbf{y}) - \mathbf{f}(\mathbf{w}, \hat{\mathbf{y}}) \quad [11.17]$$

5747 In this case, the argmax requires a maximization over all dependency trees for the sen-
5748 tence, which can be computed using the algorithms described in § 11.2.1. We can apply
5749 all the usual tricks from § 2.2: weight averaging, large-margin, and regularization. Mc-
5750 Donald et al. (2005,?) were the first to treat dependency parsing as a structure prediction
5751 problem, using MIRA, an online margin-based learning algorithm. Neural arc scores can
5752 be learned in the same way, backpropagating from a margin loss to updates on the feed-
5753 forward network that computes the score for each edge.

A conditional random field for arc-factored dependency parsing is built on the probability model,

$$p(\mathbf{y} \mid \mathbf{w}) = \frac{\exp \sum_{(i \rightarrow j, r) \in \mathbf{y}} \psi(i \rightarrow j, r)}{\sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp \sum_{(i' \rightarrow j', r') \in \mathbf{y}'} \psi(i' \rightarrow j', r')}.$$
 [11.18]

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 $p((i \rightarrow j, r) \mid \mathbf{w}; \theta)$, which in this case are probabilities over individual edges. 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 in the shift-reduce parser from the previous chapter (§ 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

⁴Transition systems also exist for non-projective dependency parsing (e.g., Nivre, 2008)

5785 there is only a single structure on the stack. The sequence of actions that led to the anal-
 5786 ysis is known as the **derivation**. One problem with transition-based systems is that there
 5787 may be multiple derivations for a single parse structure — a phenomenon known as **spu-**
 5788 **rious ambiguity**.

5789 11.3.1 Transition systems for dependency parsing

5790 A **transition system** consists of a representation for describing configurations of the parser,
 5791 and a set of transition actions, which manipulate the configuration. There are two main
 5792 transition systems for dependency parsing: **arc-standard**, which is closely related to shift-
 5793 reduce, and **arc-eager**, which adds an additional action that can simplify derivations (Ab-
 5794 ney and Johnson, 1991). In both cases, transitions are between **configurations** that are
 5795 represented as triples, $C = (\sigma, \beta, A)$, where σ is the stack, β is the input buffer, and A is
 5796 the list of arcs that have been created (Nivre, 2008). In the initial configuration,

$$C_{\text{initial}} = ([\text{ROOT}], w, \emptyset), \quad [11.19]$$

5797 indicating that the stack contains only the special root node ROOT, the entire input is on
 5798 the buffer, and the set of arcs is empty. An accepting configuration is,

$$C_{\text{accept}} = ([\text{ROOT}], \emptyset, A), \quad [11.20]$$

5799 where the stack contains only ROOT, the buffer is empty, and the arcs A define a spanning
 5800 tree over the input. The arc-standard and arc-eager systems define a set of transitions
 5801 between configurations, which are capable of transforming an initial configuration into
 5802 an accepting configuration. In both of these systems, the number of actions required to
 5803 parse an input grows linearly in the length of the input, making transition-based parsing
 5804 considerably more efficient than graph-based methods.

5805 11.3.1.1 Arc-standard

5806 The **arc-standard** transition system is closely related to shift-reduce, and to the LR algo-
 5807 rithm that is used to parse programming languages. It includes the following actions:

- 5808 • SHIFT: move the first item from the input buffer on to the top of the stack,

$$(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A), \quad [11.21]$$

5809 where we write $i|\beta$ to indicate that i is the leftmost item in the input buffer, and $\sigma|i$
 5810 to indicate the result of pushing i on to stack σ .

- 5811 • ARC-LEFT: create a new left-facing arc between the item on the top of the stack and
 5812 the first item in the input buffer. The head of this arc is j , which remains at the front
 5813 of the input buffer. The arc $i \leftarrow j$ is added to A . Formally,

$$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \oplus (j, i, r)), \quad [11.22]$$

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*.

5814 where r is the label of the dependency arc, and \oplus concatenates the new arc (j, i, r)
 5815 to the list A .

- 5816 • ARC-RIGHT: creates a new right-facing arc between the item on the top of the stack
 5817 and the first item in the input buffer. The head of this arc is i , which is “popped”
 5818 from the stack and pushed to the front of the input buffer. The arc $i \rightarrow j$ is added to
 5819 A . Formally,

$$(\sigma | i, j | \beta, A) \Rightarrow (\sigma, i | \beta, A \oplus (i, j, r)), \quad [11.23]$$

5820 where again r is the label of the dependency arc.

5821 Each action has preconditions. The SHIFT action can be performed only when the buffer
 5822 has at least one element. The ARC-LEFT action cannot be performed when the root node
 5823 ROOT is on top of the stack, since this node must be the root of the entire tree. The ARC-
 5824 LEFT and ARC-RIGHT remove the modifier words from the stack (in the case of ARC-LEFT)
 5825 and from the buffer (in the case of ARC-RIGHT), so it is impossible for any word to have
 5826 more than one parent. Furthermore, the end state can only be reached when every word is
 5827 removed from the buffer and stack, so the set of arcs is guaranteed to constitute a spanning
 5828 tree. An example arc-standard derivation is shown in Table 11.2.

5829 11.3.1.2 Arc-eager dependency parsing

5830 In the arc-standard transition system, a word is completely removed from the parse once
 5831 it has been made the modifier in a dependency arc. At this time, any dependents of
 5832 this word must have already been identified. Right-branching structures are common in
 5833 English (and many other languages), with words often modified by units such as prepo-
 5834 sitional phrases to their right. In the arc-standard system, this means that we must first

shift all the units of the input onto the stack, and then work backwards, creating a series of arcs, as occurs in Table 11.2. Note that the decision to shift *bagels* onto the stack guarantees that the prepositional phrase *with lox* will attach to the noun phrase, and that this decision must be made before the prepositional phrase is itself parsed. This has been argued to be cognitively implausible (Abney and Johnson, 1991); from a computational perspective, it means that a parser may need to look several steps ahead to make the correct decision.

Arc-eager dependency parsing changes the ARC-RIGHT action so that right dependents can be attached before all of their dependents have been found. Rather than removing the modifier from both the buffer and stack, the ARC-RIGHT action pushes the modifier on to the stack, on top of the head. Because the stack can now contain elements that already have parents in the partial dependency graph, two additional changes are necessary:

- A precondition is required to ensure that the ARC-LEFT action cannot be applied when the top element on the stack already has a parent in A .
- A new REDUCE action is introduced, which can remove elements from the stack if they already have a parent in A :

$$(\sigma|i, \beta, A) \Rightarrow (\sigma, \beta, A). \quad [11.24]$$

As a result of these changes, it is now possible to create the arc *like* → *bagels* before parsing the prepositional phrase *with lox*. Furthermore, this action does not imply a decision about whether the prepositional phrase will attach to the noun or verb. Noun attachment is chosen in Table 11.3, but verb attachment could be achieved by applying the REDUCE action at step 5 or 7.

11.3.1.3 Projectivity

The arc-standard and arc-eager transition systems are guaranteed to produce projective dependency trees, because all arcs are between the word at the top of the stack and the left-most edge of the buffer (Nivre, 2008). Non-projective transition systems can be constructed by adding actions that create arcs with words that are second or third in the stack (Attardi, 2006), or by adopting an alternative configuration structure, which maintains a list of all words that do not yet have heads (Covington, 2001). In **pseudo-projective dependency parsing**, a projective dependency parse is generated first, and then a set of graph transformation techniques are applied, producing non-projective edges (Nivre and Nilsson, 2005).

11.3.1.4 Beam search

In “greedy” transition-based parsing, the parser tries to make the best decision at each configuration. This can lead to search errors, when an early decision locks the parser into

σ	β	action	arc added to \mathcal{A}
1. [ROOT]	<i>they like bagels with lox</i>	SHIFT	
2. [ROOT, <i>they</i>]	<i>like bagels with lox</i>	ARC-LEFT	(<i>they</i> \leftarrow <i>like</i>)
3. [ROOT]	<i>like bagels with lox</i>	ARC-RIGHT	(ROOT \rightarrow <i>like</i>)
4. [ROOT, <i>like</i>]	<i>bagels with lox</i>	ARC-RIGHT	(<i>like</i> \rightarrow <i>bagels</i>)
5. [ROOT, <i>like</i> , <i>bagels</i>]	<i>with lox</i>	SHIFT	
6. [ROOT, <i>like</i> , <i>bagels</i> , <i>with</i>]	<i>lox</i>	ARC-LEFT	(<i>with</i> \leftarrow <i>lox</i>)
7. [ROOT, <i>like</i> , <i>bagels</i>]	<i>lox</i>	ARC-RIGHT	(<i>bagels</i> \rightarrow <i>lox</i>)
8. [ROOT, <i>like</i> , <i>bagels</i> , <i>lox</i>]	\emptyset	REDUCE	
9. [ROOT, <i>like</i> , <i>bagels</i>]	\emptyset	REDUCE	
10. [ROOT, <i>like</i>]	\emptyset	REDUCE	
11. [ROOT]	\emptyset	DONE	

Table 11.3: Arc-eager derivation of the unlabeled dependency parse for the input *they like bagels with lox*.

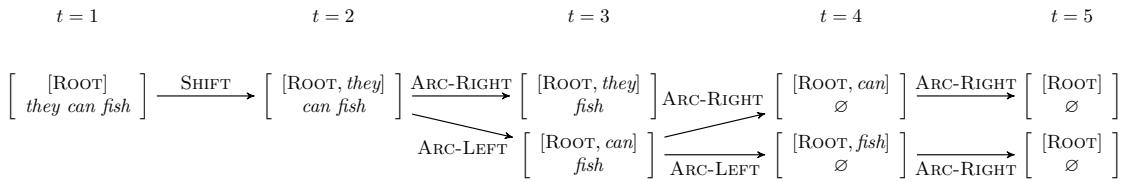


Figure 11.7: Beam search for unlabeled dependency parsing, with beam size $K = 2$. The arc lists for each configuration are not shown, but can be computed from the transitions.

5869 a poor derivation. For example, in Table 11.2, if ARC-RIGHT were chosen at step 4, then
 5870 the parser would later be forced to attach the prepositional phrase *with lox* to the verb
 5871 *likes*. Note that the *likes* \rightarrow *bagels* arc is indeed part of the correct dependency parse, but
 5872 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,

$$h_t^{(k)} = (\psi_t^{(k)}, A_t^{(k)}) \quad [11.25]$$

5873 Each hypothesis is then “expanded” by considering the set of all valid actions $\mathcal{A}(c_t^{(k)})$.
 5874 This yields a large set of new hypotheses. For each new hypothesis $A_t^{(k)} \oplus a$, we compute
 5875 an updated score. The top k hypotheses by this scoring metric are kept, and parsing
 5876 proceeds to the next step (Zhang and Clark, 2008). Note that beam search requires a

5877 scoring function which applies to action *sequences*, rather than individual actions. This
 5878 issue will be revisited in the next section.

5879 An example of beam search is shown in Figure 11.7, with a beam size of $K = 2$. For the
 5880 first transition, the only valid action is SHIFT, so there is only one possible configuration
 5881 at $t = 2$. From this configuration, there are three possible actions. The top two are ARC-
 5882 RIGHT and ARC-LEFT, and so the resulting hypotheses from these actions are on the beam
 5883 at $t = 3$. From these configurations, there are three possible actions each, but the best
 5884 two are expansions of the bottom hypothesis at $t = 3$. Parsing continues until $t = 5$, at
 5885 which point both hypotheses reach an accepting state. The best-scoring hypothesis is then
 5886 selected as the parse.

5887 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,

$$\hat{a} = \operatorname{argmax}_{a \in \mathcal{A}(c)} \psi(a, c, \mathbf{w}; \boldsymbol{\theta}), \quad [11.26]$$

5888 where $\mathcal{A}(c)$ is the set of admissible actions in the current configuration c , \mathbf{w} is the input,
 5889 and ψ is a scoring function with parameters $\boldsymbol{\theta}$ (Yamada and Matsumoto, 2003).

5890 A feature-based score can be computed, $\psi(a, c, \mathbf{w}) = \boldsymbol{\theta} \cdot \mathbf{f}(a, c, \mathbf{w})$, using features that
 5891 may consider any aspect of the current configuration and input sequence. Typical features
 5892 for transition-based dependency parsing include: the word and part-of-speech of the top
 5893 element on the stack; the word and part-of-speech of the first, second, and third elements
 5894 on the input buffer; pairs and triples of words and parts-of-speech from the top of the
 5895 stack and the front of the buffer; the distance (in tokens) between the element on the top
 5896 of the stack and the element in the front of the input buffer; the number of modifiers of
 5897 each of these elements; and higher-order dependency features as described above in the
 5898 section on graph-based dependency parsing (see, e.g., Zhang and Nivre, 2011).

5899 Parse actions can also be scored by neural networks. For example, Chen and Manning
 5900 (2014) build a feedforward networks in which the input layer consists of the concatenation
 5901 of embeddings of the same words and tags that are used in feature-based approaches:

- 5902 • the top three words on the stack, and the first three words on the buffer;
- 5903 • the first and second leftmost and rightmost children (dependents) of the top two
 5904 words on the stack;
- 5905 • the leftmost and right most grandchildren of the top two words on the stack;
- 5906 • 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,

$$\mathbf{z} = g(\Theta^{(x \rightarrow z)} \mathbf{x}(c, \mathbf{w})) \quad [11.27]$$

$$\psi(a, c, \mathbf{w}; \boldsymbol{\theta}) = \Theta_a^{(z \rightarrow y)} \mathbf{z}, \quad [11.28]$$

5907 where the vector \mathbf{z} plays the same role as the features $\mathbf{f}(a, c, \mathbf{w})$, but is a learned representation.
 5908 Chen and Manning (2014) use a cubic elementwise activation function, $g(x) = x^3$,
 5909 so that the hidden layer models products across all triples of input features. The learning
 5910 algorithm updates the embeddings as well as the parameters of the feedforward network.

5911 11.3.3 Learning to parse

5912 Transition-based dependency parsing suffers from a mismatch between the supervision,
 5913 which comes in the form of dependency trees, and the classifier’s prediction space, which
 5914 is a set of parsing actions. One solution is to create new training data by converting parse
 5915 trees into action sequences; another is to derive supervision directly from the parsing
 5916 performance.

5917 11.3.3.1 Oracle-based training

5918 A transition system can be viewed as a function from action sequences (also called **derivations**)
 5919 to parse trees. The inverse of this function is a mapping from parse trees to derivations,
 5920 which is called an **oracle**. For the arc-standard and arc-eager parsing system, an
 5921 oracle can be computed in linear time in the length of the derivation (Kübler et al., 2009,
 5922 page 32). Note that both the arc-standard and arc-eager transition systems suffer from
 5923 **spurious ambiguity**: there exist dependency parses for which multiple derivations are
 5924 possible, such as $1 \leftarrow 2 \rightarrow 3$ (Cohen et al., 2012). The oracle must choose between these
 5925 different derivations. For example, the algorithm described by Kübler et al. (2009) would
 5926 first create the left arc ($1 \leftarrow 2$), and then create the right arc, $(1 \leftarrow 2) \rightarrow 3$; another oracle
 5927 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 an classification-based objective that rewards selecting the

oracle action. For transition-based dependency parsing, maximum conditional likelihood is a typical choice (Chen and Manning, 2014; Dyer et al., 2015):

$$p(a | c, \mathbf{w}) = \frac{\exp \psi(a, c, \mathbf{w}; \boldsymbol{\theta})}{\sum_{a' \in \mathcal{A}(c)} \exp \psi(a', c, \mathbf{w}; \boldsymbol{\theta})} \quad [11.29]$$

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^N \sum_{t=1}^{|A^{(i)}|} \log p(a_t^{(i)} | c_t^{(i)}, \mathbf{w}), \quad [11.30]$$

5928 where $|A^{(i)}|$ is the length of the action sequence $A^{(i)}$.

5929 Recall that beam search requires a scoring function for action sequences. Such a score
 5930 can be obtained by adding the log-likelihoods (or hinge losses) across all actions in the
 5931 sequence (Chen and Manning, 2014).

5932 11.3.3.2 Global objectives

5933 The objective in Equation 11.30 is **locally-normalized**, in the sense that it is the prod-
 5934 uct of probabilities over individual actions. A similar characterization could be made of
 5935 non-probabilistic algorithms in which hinge-loss objectives are summed over individual
 5936 actions. In either case, training on individual actions can be sub-optimal with respect
 5937 to global performance, due to the **label bias problem** (Lafferty et al., 2001; Andor et al.,
 5938 2016).

5939 As a stylized example, suppose that a given configuration appears 100 times in the
 5940 training data, with action a_1 as the oracle action in 51 cases, and a_2 as the oracle action in
 5941 the other 49 cases. However, in cases where a_2 is correct, choosing a_1 results in a cascade
 5942 of subsequent errors, while in cases where a_1 is correct, choosing a_2 results in only a single
 5943 error. A classifier that is trained on a local objective function will learn to always choose
 5944 a_1 , but choosing a_2 would minimize the overall number of errors.

5945 This observation motivates a global objective, such as the globally-normalized condi-
 5946 tional likelihood,

$$p(A^{(i)} | \mathbf{w}; \boldsymbol{\theta}) = \frac{\exp \sum_{t=1}^{|A^{(i)}|} \psi(a_t^{(i)}, c_t^{(i)}, \mathbf{w})}{\sum_{A' \in \mathbb{A}(\mathbf{w})} \exp \sum_{t=1}^{|A'|} \psi(a'_t, c'_t, \mathbf{w})}, \quad [11.31]$$

where the denominator sums over the set of all possible action sequences, $\mathbb{A}(\mathbf{w})$.⁵ In the conditional random field model for sequence labeling (§ 7.5.3), it was possible to compute

⁵Andor et al. (2016) prove that the set of globally-normalized conditional distributions is a strict superset of the set of locally-normalized conditional distributions, and that globally-normalized conditional models are therefore strictly more expressive.

this sum explicitly, using dynamic programming. In transition-based parsing, this is not possible. However, the sum can be approximated using beam search,

$$\sum_{A' \in \mathbb{A}(\mathbf{w})} \exp \sum_{t=1}^{|A'|} \psi(a'_t, c'_t, \mathbf{w}) \approx \sum_{k=1}^K \exp \sum_{t=1}^{|A^{(k)}|} \psi(a_t^{(k)}, c_t^{(k)}, \mathbf{w}), \quad [11.32]$$

where $A^{(k)}$ is an action sequence on a beam of size K . This gives rise to the following loss function,

$$L(\boldsymbol{\theta}) = - \sum_{t=1}^{|A^{(i)}|} \psi(a_t^{(i)}, c_t^{(i)}, \mathbf{w}) + \log \sum_{k=1}^K \exp \sum_{t=1}^{|A^{(k)}|} \psi(a_t^{(k)}, c_t^{(k)}, \mathbf{w}). \quad [11.33]$$

5947 The derivatives of this loss involve expectations with respect to a probability distribution
5948 over action sequences on the beam.

5949 11.3.3.3 *Early update and the incremental perceptron

5950 When learning in the context of beam search, the goal is to learn a decision function so that
5951 the gold dependency parse is always reachable from at least one of the partial derivations
5952 on the beam. (The combination of a transition system (such as beam search) and a scoring
5953 function for actions is known as a **policy**.) To achieve this, we can make an **early update**
5954 as soon as the oracle action sequence “falls off” the beam, even before a complete analysis
5955 is available (Collins and Roark, 2004; Daumé III and Marcu, 2005). The loss can be based
5956 on the best-scoring hypothesis on the beam, or the sum of all hypotheses (Huang et al.,
5957 2012).

5958 For example, consider the beam search in Figure 11.7. In the correct parse, *fish* is the
5959 head of dependency arcs to both of the other two words. In the arc-standard system,
5960 this can be achieved only by using SHIFT for the first two actions. At $t = 3$, the oracle
5961 action sequences has fallen off the beam. The parse should therefore stop, and update the
5962 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
5963 oracle sequence, and $\{A_{1:3}^{(k)}\}$ is the beam.

5964 This integration of incremental search and learning was first developed in the **incremental**
5965 **perceptron** (Collins and Roark, 2004). This method updates the parameters with
5966 respect to a hinge loss, which compares the top-scoring hypothesis and the gold action
5967 sequence, up to the current point t . This approach can be applied to any discriminatively-
5968 trained model. Several improvements to this basic protocol are possible:

- 5969 • As noted earlier, the gold dependency parse can be derived by multiple action se-
5970 quences. Rather than checking for the presence of a single oracle action sequence on
5971 the beam, we can check if the gold dependency parse is *reachable* from the current
5972 beam, using a **dynamic oracle** (Goldberg and Nivre, 2012).

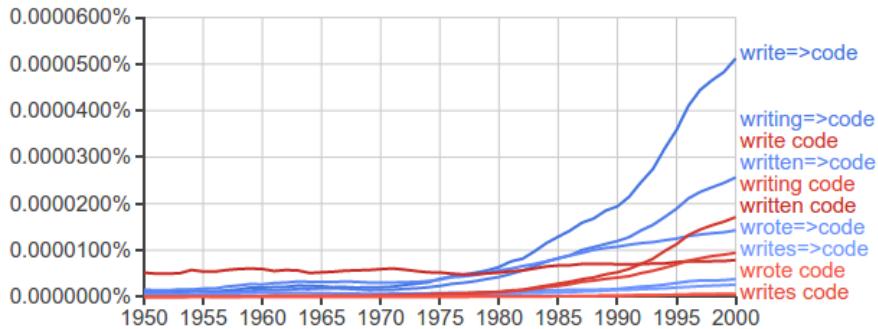


Figure 11.8: Google n-grams results for the bigram *write code* and the dependency arc *write => code* (and their morphological variants)

- 5973 • By maximizing the score of the gold action sequence, we are training a decision
- 5974 function to find the correct action given the gold context. But in reality, the parser
- 5975 will make errors, and the parser is not trained to find the best action given a context
- 5976 that may not itself be optimal. This issue is addressed by various generalizations of
- 5977 incremental perceptron, known as **learning to search** (Daumé III et al., 2009). Some
- 5978 of these methods are discussed in chapter 15.

5979 11.4 Applications

5980 Dependency parsing is used in many real-world applications: any time you want to know
 5981 about pairs of words which might not be adjacent, you can use dependency arcs instead
 5982 of regular expression search patterns. For example, we may want to match strings like
 5983 *delicious pastries*, *delicious French pastries*, and *the pastries are delicious*⁶

5984 It is possible to search the Google n-grams corpus by dependency edges, finding the
 5985 trend in how often a dependency edge appears over time. For example, we might be
 5986 interested in knowing when people started talking about *writing code*, but we also want
 5987 *write some code*, *write the code*, *write all the code*, etc. The result of a search on the dependency
 5988 edge *write → code* is shown in Figure 11.8. This capability has been applied to research in
 5989 digital humanities, such as the analysis of Shakespeare performed by Muralidharan and
 5990 Hearst (2013).

A classic application of dependency parsing is **relation extraction**, which is described

⁶Recall that the copula (in this case, *are*) is collapsed in many dependency grammars, such as the Universal Dependency treebank Nivre et al. (2016).

in chapter 17. The goal of relation extraction is to identify entity pairs, such as

(TOLSTOY, WAR AND PEACE)
 (MARQUÉZ, 100 YEARS OF SOLITUDE)
 (SHAKESPEARE, A MIDSUMMER NIGHT'S DREAM),

5991 which stand in some relation to each other (in this case, the relation is authorship). Such
 5992 entity pairs are often referenced via consistent chains of dependency relations. Therefore,
 5993 dependency paths are often a useful feature in supervised systems which learn to detect
 5994 new instances of a relation, based on labeled examples of other instances of the same
 5995 relation type (Culotta and Sorensen, 2004; Fundel et al., 2007; Mintz et al., 2009).

5996 Cui et al. (2005) show how dependency parsing can improve automated question an-
 5997 swering. Suppose you receive the following query:

5998 (11.1) What percentage of the nation's cheese does Wisconsin produce?

5999 The corpus contains this sentence:

6000 (11.2) In Wisconsin, where farmers produce 28% of the nation's cheese, ...

6001 The location of *Wisconsin* in the surface form of this string makes it a poor match for the
 6002 query. However, in the dependency graph, there is an edge from *produce* to *Wisconsin* in
 6003 both the question and the potential answer, raising the likelihood that this span of text is
 6004 relevant to the question.

6005 A final example comes from sentiment analysis. As discussed in chapter 4, the polarity
 6006 of a sentence can be reversed by negation, e.g.

6007 (11.3) *There is no reason at all to believe the polluters will suddenly become reasonable.*

6008 By tracking the sentiment polarity through the dependency parse, we can better iden-
 6009 tify the overall polarity of the sentence, determining when key sentiment words are re-
 6010 versed (Wilson et al., 2005; Nakagawa et al., 2010).

6011 11.5 Additional reading

6012 More details on dependency grammar and parsing algorithms can be found in the manuscript
 6013 by Kübler et al. (2009). For a comprehensive but whimsical overview of graph-based de-
 6014 pendency parsing algorithms, see Eisner (1997). Jurafsky and Martin (2018) describe an
 6015 **agenda-based** version of beam search, in which the beam contains hypotheses of varying
 6016 lengths. New hypotheses are added to the beam only if their score is better than the worst
 6017 item currently on the beam. Another search algorithm for transition-based parsing is

6018 **easy-first**, which abandons the left-to-right traversal order, and adds the highest-scoring
6019 edges first, regardless of where they appear (Goldberg and Elhadad, 2010). Goldberg et al.
6020 (2013) note that although transition-based methods can be implemented in linear time in
6021 the length of the input, naïve implementations of beam search will require quadratic time,
6022 due to the cost of copying each hypothesis when it is expanded on the beam. This issue
6023 can be addressed, using a more efficient data structure for the stack.

6024 Exercises

- 6025 1. Suppose you have a set of unlabeled arc scores $\psi(i \rightarrow j)$, where the score depends
6026 only on the identity of the two words. The scores include $\psi(\text{ROOT} \rightarrow j)$.
 - 6027 • Assuming each word occurs only once in the sentence ($(i \neq j) \Leftarrow (w_i \neq w_j)$),
6028 how would you construct a weighted lexicalized context-free grammar so that
6029 the score of *any* projective dependency tree is equal to the score of some equiv-
6030 alent derivation in the lexicalized context-free grammar?
 - 6031 • Verify that your method works for a simple example like *they eat fish*.
 - 6032 • How would you adapt your method to handle the case an individual word
6033 may appear multiple times in the sentence?
- 6034 2. Provide the UD-style dependency parse for the sentence *Xi-Lan eats shoots and leaves*,
6035 assuming *leaves* is a verb. Provide arc-standard and arc-eager derivations for this
6036 dependency parse.

6037

Part III

6038

Meaning

6039 Chapter 12

6040 Logical semantics

6041 A grand ambition of natural language processing is to convert natural language into a rep-
6042 resentation that supports inferences about meaning. Indeed, many potential applications
6043 of language technology involve some level of semantic understanding:

- 6044 • Answering questions, such as *where is the nearest coffeeshop?* or *what is the middle name*
6045 *of the mother of the 44th President of the United States?*.
- 6046 • Building a robot that can follow natural language instructions to execute tasks.
- 6047 • Translating a sentence from one language into another, while preserving the under-
6048 lying meaning.
- 6049 • Fact-checking an article by searching the web for contradictory evidence.
- 6050 • Logic-checking an argument by identifying contradictions, ambiguity, and unsup-
6051 ported assertions.

6052 Each of these applications can be performed by converting natural language into a
6053 **meaning representation**. To be useful, a meaning representation must meet several cri-
6054 teria:

- 6055 • **c1**: it should be unambiguous (unlike natural language);
- 6056 • **c2**: it should provide a way to link language to external knowledge, observations,
6057 and actions;
- 6058 • **c3**: it should support computational **inference**, so that meanings can be combined
6059 to derive additional knowledge;
- 6060 • **c4**: it should be expressive enough to cover the range of things that people talk about
6061 in natural language.

6062 Much more than this can be said about the question of how best to represent knowledge
 6063 for computation (e.g., Sowa, 2000), but we will focus on these criteria.

6064 12.1 Meaning and denotation

6065 The first criterion for a meaning representation is that statements in the representation
 6066 should be unambiguous — they should have only one possible interpretation. Natural
 6067 language does not have this property: as we saw in chapter 10, sentences like *cats scratch*
 6068 *people with claws* have multiple interpretations.

6069 But what does it mean for a statement to be unambiguous? Programming languages
 6070 provide a useful example: the output of a program is completely specified by the rules of
 6071 the language, and the properties of the environment in which the program is run. For ex-
 6072 ample, the python code $5 + 3$ will have the output 8, as will the codes $(4 * 4) - (3 * 3) + 1$
 6073 and $((8))$. This output is known as the **denotation** of the program, and can be written
 6074 as,

$$\llbracket 5+3 \rrbracket = \llbracket (4 * 4) - (3 * 3) + 1 \rrbracket = \llbracket ((8)) \rrbracket = 8. \quad [12.1]$$

6075 In this sense, the program's output is its "meaning".

6076 The denotations of these arithmetic expressions are determined by the meaning of the
 6077 **constants** (e.g., 5, 3) and the **relations** (e.g., $+$, $*$, $(,)$). Now let's consider another snippet
 6078 of python code, `double(4)`. The denotation of this code could be, $\llbracket \text{double}(4) \rrbracket = 8$, or
 6079 it could be $\llbracket \text{double}(4) \rrbracket = 44$ — it depends on the meaning of `double`. This meaning
 6080 is defined in a **world model** \mathcal{M} as an infinite set of pairs. We write the denotation with
 6081 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
 6082 model would also define the (infinite) list of constants, e.g., $\{0, 1, 2, \dots\}$. As long as the
 6083 denotation of string ϕ in model \mathcal{M} can be computed unambiguously, the language can be
 6084 said to be unambiguous.

6085 This approach to meaning is known as **model-theoretic semantics**, and it addresses
 6086 not only criterion *c1* (no ambiguity), but also *c2* (connecting language to external knowl-
 6087 edge, observations, and actions). For example, we can connect a representation of the
 6088 meaning of a statement like *the capital of Georgia* with a world model that includes knowl-
 6089 edge base of geographical facts, obtaining the denotation `Atlanta`. We might populate
 6090 a world model by applying an image analysis algorithm to Figure 12.1, and then use this
 6091 world model to evaluate **propositions** like *a man is riding a moose*. Another desirable prop-
 6092 erty of model-theoretic semantics is that when the facts change, the denotations change
 6093 too: the meaning representation of *President of the USA* would have a different denotation
 6094 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 [todo: the image is from 1912, so out of copyright? <https://blogs.harvard.edu/houghton/2013/09/20/myths-debunked-sadly-theodore-roosevelt-never-rode-a-moose/>]

6095 12.2 Logical representations of meaning

6096 If we can find a meaning representation which supports model-theoretic denotation, then
 6097 we will have met criteria $c1$ and $c2$. The final criterion is $c3$, which requires that the
 6098 meaning representation support inference — for example, automatically deducing new
 6099 facts from known premises. While many representations have been proposed that meet
 6100 these criteria, the most mature is the language of first-order logic.¹

6101 12.2.1 Propositional logic

6102 The bare bones of logical meaning representation are Boolean operations on propositions:

6103 **Propositional symbols** Greek symbols like ϕ and ψ will be used to represent **proposi-**
 6104 **tions**, which are statements that are either true or false. For example, ϕ may corre-
 6105 spond to the proposition, *bagels are delicious*.

¹Relevant alternatives 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).

6106 **Boolean operators** We can build up more complex propositional formulas from Boolean
 6107 operators. These include:

- 6108 • Negation $\neg\phi$, which is true if ϕ is false.
- 6109 • Conjunction, $\phi \wedge \psi$, which is true if both ϕ and ψ are true.
- 6110 • Disjunction, $\phi \vee \psi$, which is true if at least one of P and Q is true
- 6111 • Implication, $\phi \Rightarrow \psi$, which is true unless ϕ is true and ψ is false. Implication
 6112 has identical truth conditions to $\neg\phi \vee \psi$.
- 6113 • Equivalence, $\phi \Leftrightarrow \psi$, which is true if ϕ and ψ are both true or both false. Equiv-
 6114 alence has identical truth conditions to $(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$.

6115 It is not strictly necessary to have all five Boolean operators: readers familiar with
 6116 Boolean logic will know that it is possible to construct all other operators from either
 6117 the NAND (not-and) or NOR (not-or) operators. Nonetheless, it is typical to use all five
 6118 operators above for clarity. It is possible to define a number of “laws” for these Boolean
 6119 operators, such as,

- 6120 • **commutativity:** $\phi \wedge \psi = \psi \wedge \phi$, $\phi \vee \psi = \psi \vee \phi$
- 6121 • **associativity:** $\phi \wedge (\psi \wedge \chi) = (\phi \wedge \psi) \wedge \chi$, $\phi \vee (\psi \vee \chi) = (\phi \vee \psi) \vee \chi$
- 6122 • **complementation:** $\phi \wedge \neg\phi = \perp$, $\phi \vee \neg\phi = \top$, where \top indicates a true proposition
 6123 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,

$$\begin{aligned} \phi \Rightarrow \psi & \quad \text{If the music is loud, Max can't sleep.} \\ \phi & \quad \text{The music is loud.} \end{aligned}$$

6124 we can derive ψ (*Max can't sleep*) by application of **modus ponens**, which is one of a
 6125 set of **inference rules** that can be derived from more basic laws and used to manipulate
 6126 propositional formulas. **Automated theorem provers** are capable of applying inference
 6127 rules to a set of premises to derive desired propositions (Loveland, 2016).

6128 12.2.2 First-order logic

6129 Propositional logic is so named because it treats propositions as its base units. However,
 6130 we would also like to reason about the content of the propositions themselves. For exam-
 6131 ple,

6132 (12.1) If anyone is making noise, then Max can't sleep.

6133 (12.2) Abigail is making noise.

6134 To understand the relationship between the statement *anyone is making noise* and the state-
 6135 ment *Abigail is making noise*, we need some additional formal machinery. This is provided
 6136 by **first-order logic** (FOL).

6137 In FOL, logical propositions can be constructed from relationships between entities.
 6138 Specifically, FOL extends propositional logic with the following classes of terms:

6139 **Constants** These are elements that name individual entities in the model, such as MAX
 6140 and ABIGAIL. The **denotation** of each constant in a model \mathcal{M} is an element in the
 6141 model, e.g., $[\![\text{MAX}]\!] = m$ and $[\![\text{ABIGAIL}]\!] = a$.

6142 **Relations** Relations can be thought of as sets of entities, or sets of tuples. For example,
 6143 the relation CAN-SLEEP is defined as the set of entities who can sleep, and has the
 6144 denotation $[\![\text{CAN-SLEEP}]\!] = \{a, m, \dots\}$. We can test the truth value of the proposition
 6145 CAN-SLEEP(MAX) by asking whether $[\![\text{MAX}]\!] \in [\![\text{CAN-SLEEP}]\!]$. Logical relations that
 6146 are defined over sets of entities are sometimes called **properties**.

6147 Relations may also be ordered tuples of entities. For example BROTHER(MAX,ABIGAIL)
 6148 expresses the proposition that MAX is the brother of ABIGAIL. The denotation of
 6149 such relations is a set of tuples, $[\![\text{BROTHER}]\!] = \{ (m, a), (x, y), \dots \}$. We can test the
 6150 truth value of the proposition BROTHER(MAX,ABIGAIL) by asking whether the tuple
 6151 $([\![\text{MAX}]\!], [\![\text{ABIGAIL}]\!])$ is in the denotation $[\![\text{BROTHER}]\!]$.

We can now express statements like *Max can't sleep* and *Max is Abigail's brother*:

$$\neg \text{CAN-SLEEP}(\text{MAX}) \\ \text{BROTHER}(\text{MAX}, \text{ABIGAIL}).$$

We can also combine these statements using Boolean operators, such as,

$$(\text{BROTHER}(\text{MAX}, \text{ABIGAIL}) \vee \text{BROTHER}(\text{MAX}, \text{STEVE})) \Rightarrow \neg \text{CAN-SLEEP}(\text{MAX}).$$

6152 We have thus far described a fragment of first-order logic, which permits only state-
 6153 ments about specific entities. To support inferences about statements like *If anyone is*
 6154 *making noise, then Max can't sleep*, we will need two additional elements in the meaning
 6155 representation:

6156 **Variables** These are mechanisms for referring to entities that are not locally specified. We
 6157 can then write $\text{CAN-SLEEP}(x)$ or $\text{BROTHER}(x, \text{ABIGAIL})$. In these cases, x is an **free**
 6158 **variable**, meaning that we have not committed to any particular assignment.

6159 **Quantifiers** Variables are bound by quantifiers. There are two quantifiers in first-order
 6160 logic.²

- 6161 • The **existential quantifier** \exists , which indicates that there must be at least one en-
 6162 tity to which the variable can refer. For example, the statement $\exists x \text{MAKES-NOISE}(x)$
 6163 indicates that there is at least one entity for which MAKES-NOISE is true.
 6164 • The **universal quantifier** \forall , which indicates that the the variable must be able
 6165 to refer to any entity. For example, the statement,

$$\neg \text{MAKES-NOISE}(\text{ABIGAIL}) \Rightarrow (\forall x \text{CAN-SLEEP}(x)) \quad [12.3]$$

6166 asserts that every entity can sleep if Abigail does not make noise.

6167 The expressions $\exists x$ and $\forall x$ make x into a **bound variable**. A formula that contains
 6168 no free variables is a **sentence**.

6169 **Functions** Functions map from entities to entities, e.g., $\llbracket \text{CAPITAL-OF(GEORGIA)} \rrbracket = \llbracket \text{ATLANTA} \rrbracket$.
 6170 With functions, we also add an equality operator, so that it is possible to make state-
 6171 ments like,

$$\forall x \exists y \text{MOTHER-OF}(x) = \text{DAUGHTER-OF}(y). \quad [12.4]$$

6172 Note that MOTHER-OF is a functional analogue of the relation MOTHER, so that
 6173 $\text{MOTHER-OF}(x) = y$ if $\text{MOTHER}(x, y)$. Any logical formula that uses functions can be
 6174 rewritten using only relations and quantification. For example,

$$\text{MAKES-NOISE}(\text{MOTHER-OF}(\text{ABIGAIL})) \quad [12.5]$$

6175 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:

$$\forall x \exists y \text{ SPEAKS}(x, y) \quad [12.6]$$

$$\exists y \forall x \text{ SPEAKS}(x, y). \quad [12.7]$$

6176 In the first case, y may refer to several different languages, while in the second case, there
 6177 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*,

$$\forall P \forall x ((\text{GOOD-BOXER}(x) \Rightarrow P(x)) \Rightarrow P(\text{BUTCH})) \quad [12.2]$$

This example is from Blackburn and Bos (2005), who also show how first-order logic can “approximate” second-order and higher-order logics, by reifying sets as additional entities in the model.

6178 **12.2.2.1 Truth-conditional semantics**

6179 One way to look at the meaning of an FOL sentence ϕ is as a set of **truth conditions**, or
 6180 models under which ϕ is satisfied. But how can we determine whether a sentence in first-
 6181 order logic is true or false? We will approach this inductively, starting with a predicate
 6182 applied to a tuple of constants. The truth of such a sentence depends on whether the
 6183 tuple of denotations of the constants is in the denotation of the predicate. For example,
 6184 CAPITAL(GEORGIA,ATLANTA) is true in model \mathcal{M} iff,

$$(\llbracket \text{GEORGIA} \rrbracket_{\mathcal{M}}, \llbracket \text{ATLANTA} \rrbracket_{\mathcal{M}}) \in \llbracket \text{CAPITAL} \rrbracket_{\mathcal{M}}. \quad [12.8]$$

6185 The Boolean operators \wedge, \vee, \dots provide ways to construct more complicated sentences,
 6186 and the truth of such statements can be assessed based on the truth tables associated with
 6187 these operators. The statement $\exists x\phi$ is true if there is some assignment of the variable x
 6188 to an entity in the model such that ϕ is true; the statement $\forall x\phi$ is true if ϕ is true under
 6189 all possible assignments of x . More formally, we would say that ϕ is **satisfied** under \mathcal{M} ,
 6190 written as $\mathcal{M} \models \phi$.

6191 Truth conditional semantics allows us to define several other properties of sentences
 6192 and pairs of sentences. Suppose that in every \mathcal{M} under which ϕ is satisfied, another
 6193 formula ψ is also satisfied; then ϕ **entails** ψ , sometimes written $\phi \models \psi$ [todo: double
 6194 check]. For example,

$$\text{CAPITAL(GEORGIA,ATLANTA)} \models \exists x \text{CAPITAL(GEORGIA, } x\text{)}. \quad [12.9]$$

6195 A statement that is satisfied under any model, such as $\phi \vee \neg\phi$, is **valid**; a statement that is
 6196 not satisfied under any model, such as $\phi \wedge \neg\phi$, is **unsatisfiable**, or **inconsistent**. A **model**
 6197 **checker** is a program that determines whether a sentence ϕ is satisfied in \mathcal{M} . A **model**
 6198 **builder** is a program that constructs a model in which ϕ is satisfied. The problems of
 6199 checking for consistency and validity in first-order logic are **undecidable**, meaning that
 6200 there is no algorithm that can automatically determine whether an FOL formula is valid
 6201 or inconsistent.

6202 **12.2.2.2 Inference in first-order logic**

6203 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
 6204 can now represent such statements in first-order logic, but how are we to perform the
 6205 inference that *Max can't sleep*? One approach is to use “generalized” versions of proposi-
 6206 tional inference rules like modus ponens, which can be applied to FOL formulas. By
 6207 repeatedly applying such inference rules to a knowledge base of facts, it is possible to
 6208 produce proofs of desired propositions. To find the right sequence of inferences to derive

6210 a desired theorem, classical artificial intelligence search algorithms like backward chaining
 6211 can be applied. Such algorithms are implemented in interpreters for the `prolog` logic
 6212 programming language (Pereira and Shieber, 2002).

6213 12.3 Semantic parsing and the lambda calculus

6214 The previous section has laid out a lot of formal machinery, which we will now try to
 6215 unite with natural language. Given an English sentence like *Alex likes Brit*, how can we
 6216 obtain the desired first-order logical representation, `LIKES(ALEX,BRIT)`? This is the task of
 6217 **semantic parsing**. Just as a syntactic parser is a function from a natural language sentence
 6218 to a syntactic structure such as a phrase structure tree, a semantic parser is a function from
 6219 natural language to logical formulas.

6220 As in syntactic analysis, semantic parsing is difficult because the space of inputs and
 6221 outputs is very large, and their interaction is complex. Our best hope is that, like syntactic
 6222 parsing, semantic parsing can somehow be decomposed into simpler sub-problems. This
 6223 idea, usually attributed to the German philosopher Gottlob Frege, is called the **principle**
 6224 **of compositionality**: the meaning of a complex expression is a function of the meanings of
 6225 that expression's constituent parts. We will define these "constituent parts" as syntactic
 6226 elements like noun phrases and verb phrases. These constituents are combined using
 6227 function application: if the syntactic parse contains the production $x \rightarrow y z$, then the
 6228 semantics of x , written $x.\text{sem}$, will be computed as a function of the semantics of the
 6229 constituents, $y.\text{sem}$ and $z.\text{sem}$.³ ⁴

6230 12.3.1 The lambda calculus

6231 Let's see how this works for a simple sentence like *Alex likes Brit*, whose syntactic structure
 6232 is shown in Figure 12.2. Our goal is the formula, `LIKES(ALEX,BRIT)`, and it is clear that the
 6233 meaning of the constituents *Alex* and *Brit* should be `ALEX` and `BRIT`. That leaves two
 6234 more constituents: the verb *likes*, and the verb phrase *likes Brit*. How can we define the
 6235 meaning of these units in a way that enables us to recover the desired meaning for the
 6236 entire sentence? If the meanings of *Alex* and *Brit* are constants, then the meanings of *likes*
 6237 and *likes Brit* must be functional expressions, which can be applied to their siblings to
 6238 produce the desired analyses.

³§ 9.3.2 briefly discusses alternative syntactic formalisms, including Combinatory Categorial Grammar (CCG). 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).

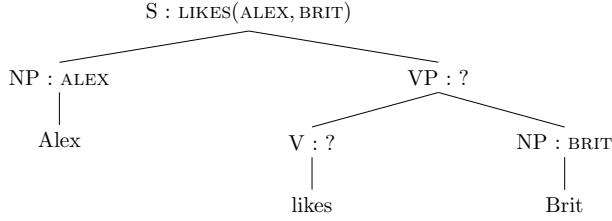


Figure 12.2: The principle of compositionality requires that we identify meanings for the constituents *likes* and *likes Brit* that will make it possible to compute the meaning for the entire sentence.

6239 Modeling these partial analyses requires extending the first-order logic meaning rep-
 6240 resentation. We do this by adding **lambda expressions**, which are descriptions of anonym-
 6241 ous functions,⁵ e.g.,

$$\lambda x.\text{LIKES}(x, \text{BRIT}). \quad [12.10]$$

6242 We take this functional expression to be the meaning of the verb phrase *likes Brit*; it takes a
 6243 single argument, and returns the result of substituting that argument for x in the expres-
 6244 sion $\text{LIKES}(x, \text{BRIT})$. We write this substitution as,

$$(\lambda x.\text{LIKES}(x, \text{BRIT}))@\text{ALEX} = \text{LIKES}(\text{ALEX}, \text{BRIT}), \quad [12.11]$$

6245 with the symbol “@” indicating function application. Function application in the lambda
 6246 calculus is sometimes called **β -reduction** or **β -conversion**. We will write $\phi@\psi$ to indicate
 6247 a function application to be performed by β -reduction, and $\phi(\psi)$ to indicate a function or
 6248 predicate in the final logical form.

6249 Equation 12.11 shows how to obtain the desired semantics for the sentence *Alex likes*
 6250 *Brit*: by applying the lambda expression $\lambda x.\text{LIKES}(x, \text{BRIT})$ to the logical constant *ALEX*.
 6251 This rule of composition can be specified in a syntactic-semantic grammar: for the syntac-
 6252 tic production $S \rightarrow \text{NP VP}$, we have the semantic rule $\text{VP.sem}@\text{NP.sem}$.

The meaning of the meaning of the transitive verb phrase can also be obtained by function application on its syntactic constituents. For the syntactic production $\text{VP} \rightarrow \text{V NP}$, we apply the semantic rule,

$$\text{VP.sem} = (\text{V.sem})@\text{NP.sem} \quad [12.12]$$

$$= (\lambda y.\lambda x.\text{LIKES}(x, y)) @ (\text{BRIT}) \quad [12.13]$$

$$= \lambda x.\text{LIKES}(x, \text{BRIT}). \quad [12.14]$$

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.

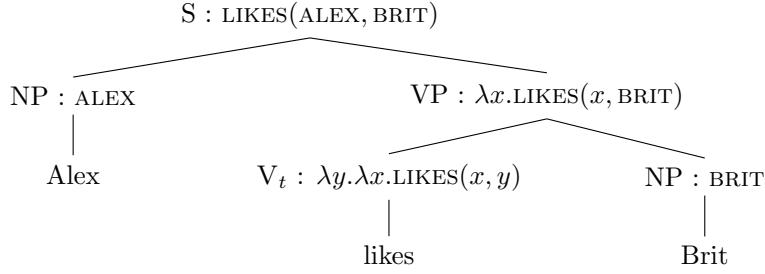


Figure 12.3: Derivation of the semantic representation for *Alex likes Brit* in the grammar G_1 .

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

6253 Here we have defined the meaning of the transitive verb *likes* as a lambda expression
 6254 whose output is **another** lambda expression: it takes y as an argument to fill in one of
 6255 the slots in the LIKES relation, and returns a lambda expression that is ready to take an
 6256 argument to fill in the other slot.⁶

6257 Table 12.1 shows a minimal syntactic/semantic grammar fragment, which we will call
 6258 G_1 . The complete **derivation** of *Alex likes Brit* in G_1 is shown in Figure 12.3. In addition to
 6259 the transitive verb *likes*, the grammar also includes the intransitive verb *sleeps*; it should be
 6260 clear how to derive the meaning of sentences like *Alex sleeps*. For verbs that can be either
 6261 transitive or intransitive, such as *eats*, we would have two terminal productions, one for
 6262 each sense (terminal productions are also called the **lexical entries**). Indeed, most of the
 6263 grammar is in the **lexicon** (the terminal productions), since these productions select the
 6264 basic units of the semantic interpretation.

⁶This can be written in a few different ways. More informally, we can write $\lambda y. x. \text{LIKES}(x, y)$, indicating a lambda expression that takes two arguments; this would be acceptable in functional programming. More formally, logicians (e.g., Carpenter, 1997) often write $\lambda y. \lambda x. \text{LIKES}(x)(y)$, indicating that each lambda expression takes exactly one argument.

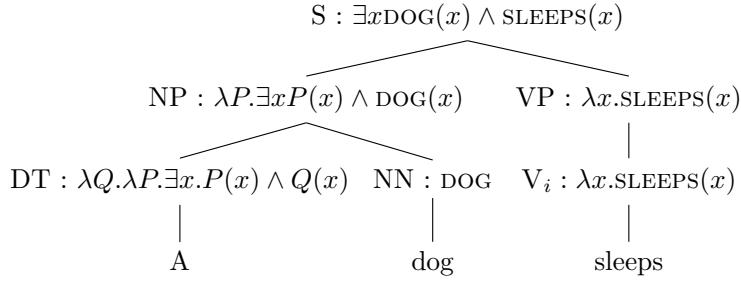


Figure 12.4: Derivation of the semantic representation for *A dog sleeps*, in grammar G_2

6265 12.3.2 Quantification

6266 Things get more complicated when we move from sentences about named entities to sen-
 6267 tences that involve more general noun phrases. Let's consider the example, *A dog sleeps*,
 6268 which has the meaning $\exists x \text{DOG}(x) \wedge \text{SLEEPS}(x)$. Clearly, the DOG relation will be intro-
 6269 duced by the word *dog*, and the SLEEP relation will be introduced by the word *sleeps*.⁷
 6270 The existential quantifier \exists must be introduced by the lexical entry for the determiner *a*.⁷
 6271 However, this seems problematic for the compositional approach taken in the grammar
 6272 G_1 : if the semantics of the noun phrase *a dog* is an existentially quantified expression, how
 6273 can it be the argument to the semantics of the verb *sleeps*, which expects an entity? And
 6274 where does the logical conjunction come from?

6275 There are a few different approaches to handling these issues.⁸ We will begin by re-
 6276 versing the semantic relationship between subject NPs and VPs, so that the production
 6277 $S \rightarrow \text{NP VP}$ has the semantics $\text{NP.sem} @ \text{VP.sem}$: the meaning of the sentence is now the
 6278 semantics of the noun phrase applied to the verb phrase. The implications of this change
 6279 are best illustrated by exploring the derivation of the example, shown in Figure 12.4. Let's
 6280 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,

$$\text{NP.sem} = \text{DET.sem} @ \text{NN.sem} \quad [12.16]$$

$$= (\lambda P. \lambda Q. \exists x P(x) \wedge Q(x)) @ (\text{DOG}) \quad [12.17]$$

$$= \lambda Q. \exists x \text{DOG}(x) \wedge Q(x). \quad [12.18]$$

6281 This is a lambda expression that is expecting another relation, Q , which will be provided
 6282 by the verb phrase, SLEEPS. This gives the desired analysis, $\exists x \text{DOG}(x) \wedge \text{SLEEPS}(x)$.⁹

6283 If noun phrases like *a dog* are interpreted as lambda expressions, then proper nouns
 6284 like *Alex* must be treated in the same way. This is achieved by **type-raising** from con-
 6285 stants to lambda expressions, $x \Rightarrow \lambda P. P(x)$. After type-raising, the semantics of *Alex* is
 6286 $\lambda P. P(\text{ALEX})$ — a lambda expression that expects a relation to tell us something about
 6287 *ALEX*.¹⁰ Make sure you see how the analysis in Figure 12.4 can be applied to the sentence
 6288 *Alex sleeps.*

6289 To handle direct objects, we will perform the same type-raising operation on transitive
 6290 verbs: the meaning of verbs such as *likes* will be raised to,

$$\lambda P. \lambda x. P(\lambda y. \text{LIKES}(x, y)) \quad [12.19]$$

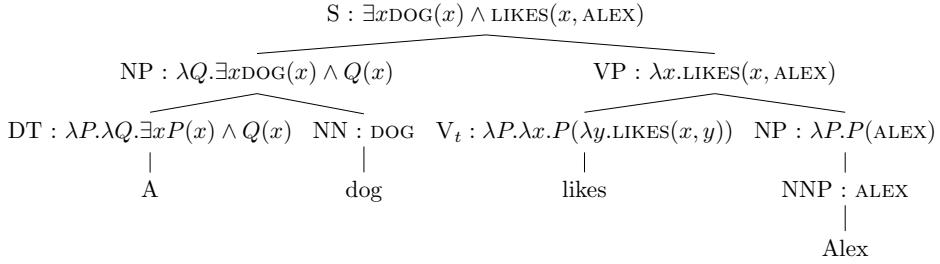
As a result, we can keep the verb phrase production $\text{VP.sem} = \text{V.sem} @ \text{NP.sem}$, knowing
 that the direct object will provide the function P in Equation 12.19. To see how this works,
 let's analyze the verb phrase *likes a dog*. After uniquely relabeling each lambda variable,
 we have,

$$\begin{aligned} \text{VP.sem} &= \text{V.sem} @ \text{NP.sem} \\ &= (\lambda P. \lambda x. P(\lambda y. \text{LIKES}(x, y))) @ (\lambda Q. \exists z \text{DOG}(z) \wedge Q(z)) \\ &= \lambda x. (\lambda Q. \exists z \text{DOG}(z) \wedge Q(z)) @ (\lambda y. \text{LIKES}(x, y)) \\ &= \lambda x. \exists z \text{DOG}(z) \wedge (\lambda y. \text{LIKES}(x, y)) @ z \\ &= \lambda x. \exists z \text{DOG}(z) \wedge \text{LIKES}(x, z). \end{aligned}$$

6291 These changes are summarized in the revised grammar G_2 , shown in Table 12.2. Fig-
 6292 ure 12.5 shows a derivation that involves a transitive verb, an indefinite noun phrase, and
 6293 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 (e, t) . A transitive verb has type

Figure 12.5: Derivation of the semantic representation for *A dog likes Alex*.

S	\rightarrow NP VP	NP.sem@VP.sem
VP	\rightarrow V _t NP	V _t .sem@NP.sem
VP	\rightarrow V _i	V _i .sem
NP	\rightarrow DET NN	DET.sem@NN.sem
NP	\rightarrow NNP	$\lambda P. P(\text{NNP.sem})$
DET	$\rightarrow a$	$\lambda P. \lambda Q. \exists x P(x) \wedge Q(x)$
DET	\rightarrow every	$\lambda P. \lambda Q. \forall x (P(x) \Rightarrow Q(x))$
V _t	\rightarrow likes	$\lambda P. \lambda x. P(\lambda y. \text{LIKES}(x, y))$
V _i	\rightarrow sleeps	$\lambda x. \text{SLEEPS}(x)$
NN	\rightarrow dog	DOG
NNP	\rightarrow Alex	ALEX
NNP	\rightarrow Brit	BRIT

Table 12.2: G_2 , a syntactic/semantic context-free grammar fragment, which supports quantified noun phrases

6294 12.4 Learning semantic parsers

6295 As with syntactic parsing, any syntactic/semantic grammar with sufficient coverage will
 6296 **overgenerate**, producing many possible analyses for any given sentence. Machine learning
 6297 is the dominant approach to selecting a single analysis. We will focus on algorithms
 6298 that learn to score logical forms by attaching weights to features of their derivations (Zettle-
 6299 moyer and Collins, 2005). Alternative approaches include transition-based parsing (Zelle
 6300 and Mooney, 1996; Misra and Artzi, 2016) and methods inspired by machine transla-
 6301 tion (Wong and Mooney, 2006). Methods also differ in the form of supervision used for

($e, (e, t)$): 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 $((e, t), t)$: it expects a function from entities to truth values, and returns a truth value.

learning, 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 geographical 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 (Bollicker 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

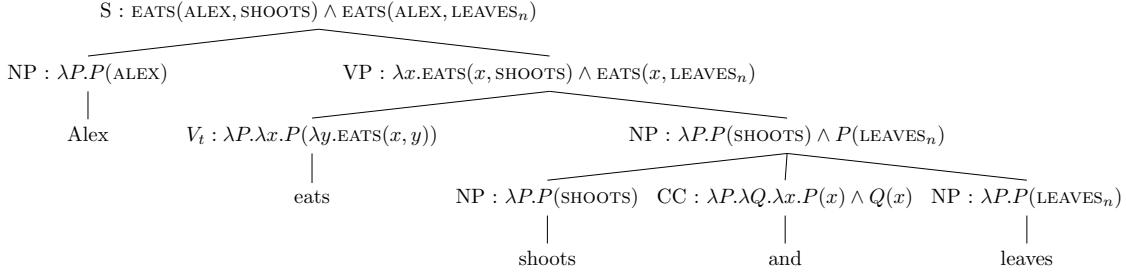
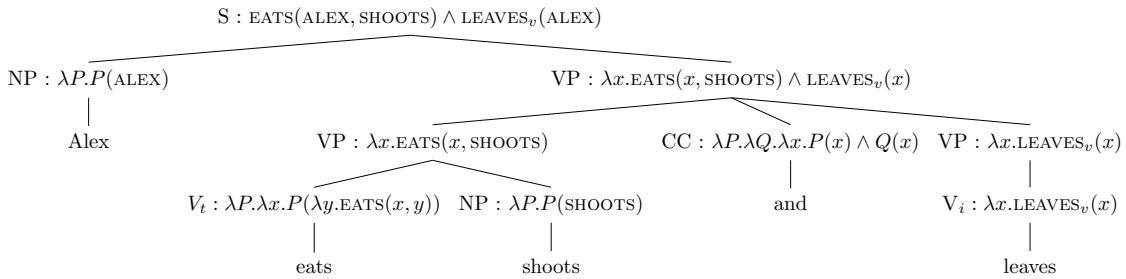
Let $w^{(i)}$ indicate a sequence of text, and let $y^{(i)}$ indicate the desired logical form. For example:

$$\begin{aligned} w^{(i)} &= \text{Alex eats shoots and leaves} \\ y^{(i)} &= \text{EATS(ALEX,SHOOTS)} \wedge \text{EATS(ALEX,LEAVES)} \end{aligned}$$

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.)

Let us introduce an additional variable z , representing the **derivation** of the logical form y from the text w . We assume that the feature function decomposes across the

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*

6333 productions in the derivation, $f(\mathbf{w}, \mathbf{z}, \mathbf{y}) = \sum_{t=1}^T f(\mathbf{w}, z_t, \mathbf{y})$, where z_t indicates a single syntactic/semantic production. For example, we might have a feature for the high-level production $S \rightarrow NP VP : NP.sem@VP.sem$, as well as for terminal productions like $NNP \rightarrow Alex : ALEX$. Under this decomposition, we can compute scores for each semantically-annotated subtree in the analysis of \mathbf{w} , and can therefore apply bottom-up parsing algorithms like CKY (§ 10.1) to find the best-scoring semantic analysis.

6339 Figure 12.6 shows a derivation of the correct semantic analysis of the sentence *Alex*
 6340 *eats shoots and leaves*, in a simplified grammar in which the plural noun phrases *shoots*
 6341 and *leaves* are interpreted as logical constants *SHOOTS* and *LEAVES_n*. Figure 12.7 shows a
 6342 derivation of an incorrect analysis. Assuming one feature per production, the perceptron
 6343 update is shown in Table 12.3. From this update, the parser would learn to prefer the noun
 6344 interpretation of *leaves* over the verb interpretation. It would also learn to prefer noun
 6345 phrase coordination over verb phrase coordination. Note that we could easily replace the
 6346 perceptron with a conditional random field. In this case, the online updates would be
 6347 based on feature expectations, which can be computed using bottom-up algorithms like
 6348 inside-outside (§ 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)

6349 **12.4.2 Learning from logical forms**

Complete derivations are expensive to annotate, and are rarely available.¹¹ More recent work has focused 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,

$$p(y, z | w) = \frac{\exp(\theta \cdot f(w, z, y))}{\sum_{y', z'} \exp(\theta \cdot f(w, z', y'))}, \quad [12.20]$$

6350 which is the standard log-linear model, applied to the logical form y and the derivation
6351 z .

Since the derivation z completely 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,

$$p(y | w) = \sum_z p(y, z | w). \quad [12.21]$$

We can then have the semantic parser select the logical form with the maximum log marginal probability,

$$\log \sum_z p(y, z | w) = \log \sum_z \frac{\exp(\theta \cdot f(w, z, y))}{\sum_{y', z'} \exp(\theta \cdot f(w, z', y'))} \quad [12.22]$$

$$\propto \log \sum_z \exp(\theta \cdot f(w, z', y')) \quad [12.23]$$

$$\geq \max_z \theta \cdot f(w, z, y). \quad [12.24]$$

6352 Note that it is impossible to push the log term inside the sum over z , meaning that our
6353 usual linear scoring function does not apply. We can recover this scoring function only

¹¹An exception is the work of Ge and Mooney (2005), who annotate the meaning of each syntactic constituents for several hundred sentences.

in approximation, by taking the max (rather than the sum) over derivations z , which provides a lower bound.

Learning can be performed by maximizing the log marginal likelihood,

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^N \log p(\mathbf{y}^{(i)} | \mathbf{w}^{(i)}; \boldsymbol{\theta}) \quad [12.25]$$

$$= \sum_{i=1}^N \log \sum_z p(\mathbf{y}^{(i)}, z^{(i)} | \mathbf{w}^{(i)}; \boldsymbol{\theta}). \quad [12.26]$$

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 initialization.

The derivative of Equation 12.26 is,

$$\frac{\partial \ell_i}{\partial \boldsymbol{\theta}} = \sum_z p(z | \mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) f(\mathbf{w}, z, \mathbf{y}) - \sum_{y', z'} p(y', z' | \mathbf{w}; \boldsymbol{\theta}) f(\mathbf{w}, z', y') \quad [12.27]$$

$$= E_{z|\mathbf{y}, \mathbf{w}} f(\mathbf{w}, z, \mathbf{y}) - E_{y, z|\mathbf{w}} f(\mathbf{w}, z, \mathbf{y}) \quad [12.28]$$

Both expectations can be computed via bottom-up algorithms like inside-outside. Alternatively, we can again maximize rather than marginalize over derivations for an approximate solution. In either case, the first term of the gradient requires us to identify derivations z that are compatible with the logical form \mathbf{y} . This can be done in a bottom-up dynamic programming algorithm, by having each cell in the table $t[i, j, X]$ include the set of all possible logical forms for $X \rightsquigarrow \mathbf{w}_{i+1:j}$. The resulting table may therefore be much larger than in syntactic parsing. This can be controlled by using pruning to eliminate intermediate analyses that are incompatible with the final logical form \mathbf{y} (Zettlemoyer and Collins, 2005), or by using beam search and restricting the size of each cell to some fixed constant (Liang et al., 2013).

If we replace each expectation in Equation 12.28 with argmax and then apply stochastic gradient descent to learn the weights, we obtain the **latent variable perceptron**, a simple and general algorithm for learning with missing data. The algorithm is shown in its most basic form in Algorithm 16, but the usual tricks such as averaging and margin loss can be applied (Yu and Joachims, 2009). Aside from semantic parsing, the latent variable perceptron has been used in tasks such as machine translation (Liang et al., 2006) and named entity recognition (Sun et al., 2009). In **latent conditional random fields**, we use the full expectations rather than maximizing over the hidden variable. This model has also been employed in a range of problems beyond semantic parsing, including parse reranking (Koo and Collins, 2005) and gesture recognition (Quattoni et al., 2007).

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  $\theta$ 

```

6379 **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 denotations 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}

6380 Similarly, in a robotic control setting, the denotation of a command would be an action or
6381 sequence of actions (Artzi and Zettlemoyer, 2013). In both cases, the idea is to reward the
6382 semantic parser for choosing an analysis whose denotation is correct: the right answer to
6383 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 $\llbracket \mathbf{y} \rrbracket$ for the text $\mathbf{w}^{(i)}$ is correct. We can then learn by maximizing a conditional-likelihood objective,

$$\ell^{(i)}(\boldsymbol{\theta}) = \log \sum_{\mathbf{y}} v_i(\mathbf{y}) \times p(\mathbf{y} \mid \mathbf{w}; \boldsymbol{\theta}) \quad [12.29]$$

$$= \log \sum_{\mathbf{y}} v_i(\mathbf{y}) \times \sum_{\mathbf{z}} p(\mathbf{y}, \mathbf{z} \mid \mathbf{w}; \boldsymbol{\theta}), \quad [12.30]$$

6384 which sums over all derivations \mathbf{z} of all valid logical forms, $\{\mathbf{y} : v_i(\mathbf{y}) = 1\}$. This cor-
6385 responds to the log-probability that the semantic parser produces a logical form with a
6386 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).

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>

- 6413 2. Convert the following examples into first-order logic, using the relations CAN-SLEEP,
 6414 MAKES-NOISE, and BROTHER.
- 6415 • If Abigail makes noise, no one can sleep.
 6416 • If Abigail makes noise, someone cannot sleep.
 6417 • None of Abigail's brothers can sleep.
 6418 • If one of Abigail's brothers makes noise, Abigail cannot sleep.
- 6419 3. Extend the grammar fragment G_1 to include the ditransitive verb *teaches* and the
 6420 proper noun *Swahili*. Show how to derive the interpretation for the sentence *Alex*
 6421 *teaches Brit Swahili*, which should be $\text{TEACHES}(\text{ALEX}, \text{BRIT}, \text{SWAHILI})$. The grammar
 6422 need not be in Chomsky Normal Form. For the ditransitive verb, use NP_1 and NP_2
 6423 to indicate the two direct objects.
- 6424 4. Derive the semantic interpretation for the sentence *Alex likes every dog*, using gram-
 6425 mar fragment G_2 .
- 6426 5. Extend the grammar fragment G_2 to handle adjectives, so that the meaning of *an
 6427 angry dog* is $\lambda P. \exists x \text{DOG}(x) \wedge \text{ANGRY}(x) \wedge P(x)$. Specifically, you should supply the
 6428 lexical entry for the adjective *angry*, and you should specify the syntactic-semantic
 6429 productions $\text{NP} \rightarrow \text{DET } \text{NOM}$, $\text{NOM} \rightarrow \text{JJ } \text{NOM}$, and $\text{NOM} \rightarrow \text{NN}$.
- 6430 6. Extend your answer to the previous question to cover copula constructions with
 6431 predicative adjectives, such as *Alex is angry*. The interpretation should be $\text{ANGRY}(\text{ALEX})$.
 6432 You should add a verb phrase production $\text{VP} \rightarrow \text{V}_{\text{cop}} \text{ JJ}$, and a terminal production
 6433 $\text{V}_{\text{cop}} \rightarrow \text{is}$. Show why your grammar extensions result in the correct interpretation.
- 6434 7. In Figure 12.6 and Figure 12.7, we treat the plurals *shoots* and *leaves* as entities. Revise
 6435 G_2 so that the interpretation of *Alex eats leaves* is $\forall x. (\text{LEAF}(x) \Rightarrow \text{EATS}(\text{ALEX}, x))$, and
 6436 show the resulting perceptron update.
- 6437 8. Statements like *every student eats a pizza* have two possible interpretations, depend-
 6438 ing on quantifier scope:

$$\forall x \exists y \text{PIZZA}(y) \wedge (\text{STUDENT}(x) \Rightarrow \text{EATS}(x, y)) \quad [12.32]$$

$$\exists y \forall x \text{PIZZA}(y) \wedge (\text{STUDENT}(x) \Rightarrow \text{EATS}(x, y)) \quad [12.33]$$

6437 Explain why these interpretations really are different, and modify the grammar G_2
 6438 so that it can produce both interpretations.

- 6439 9. Derive Equation 12.27.

- 6440 10. [todo: not sure this works] Download the GeoQuery data, get some deterministic
6441 parser that overgenerates, and try to learn a reranker that selects the correct logical
6442 form.
- 6443 11. In the GeoQuery domain, give a natural language query that has multiple plausible
6444 semantic interpretations with the same denotation. List both interpretations and the
6445 denotation.

6446 **Hint:** There are many ways to do this, but one approach involves using toponyms
6447 (place names) that could plausibly map to several different entities in the model.

6448

Chapter 13

6449

Predicate-argument semantics

6450 In this chapter, we consider more “lightweight” semantic representations. These semantic
6451 representations discard some aspects of first-order predicate calculus, but focus on
6452 predicate-argument structures. Let’s start with an example sentence:

6453 (13.1) Asha gives Boyang a book.

6454 The predicate calculus representation of this sentence would be written,

$$\exists x. \text{BOOK}(x) \wedge \text{GIVE}(\text{Asha}, \text{Boyang}, x) \quad [13.1]$$

6455 In this representation, we define variable x for the book, and we link the strings *Asha*
6456 and *Boyang* to entities Asha and Boyang. Because the action of giving involves a giver, a
6457 recipient, and a gift, the predicate GIVE must take three arguments.

6458 Now suppose we have additional information about the event, such as,

6459 (13.2) Yesterday, Asha reluctantly gave Boyang a book.

6460 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 ,

$$\begin{aligned} \exists e, x. & \text{GIVE-EVENT}(e) \wedge \text{GIVER}(e, \text{Asha}) \wedge \text{GIFT}(e, x) \wedge \text{BOOK}(e, x) \wedge \text{RECIPIENT}(e, \text{Boyang}) \\ & \wedge \text{TIME}(e, \text{Yesterday}) \wedge \text{MANNER}(e, \text{reluctantly}) \end{aligned}$$

6461 In this way, each argument of the event — the giver, the recipient, the gift — can be
 6462 represented with a relation of its own, linking the argument to the event e . The expression
 6463 GIVER(e , Asha) says that Asha plays the **role** of GIVER in the event. This reformulation
 6464 nicely handles the problem of optional information such as the time or manner of the
 6465 event, which are called **adjuncts**. Unlike arguments, adjuncts are not a mandatory part of
 6466 the relation, but under this representation, they can be expressed with additional logical
 6467 relations that are conjoined to the semantic interpretation of the sentence.¹

6468 The event semantic representation can be applied to nested clauses, e.g.,

6469 (13.3) Chris sees Asha pay Boyang.

This is done by using the event variable as an argument:

$$\begin{aligned} \exists e_1, e_2. & \text{SEE-EVENT}(e_1) \wedge \text{SEER}(e_1, \text{Chris}) \wedge \text{SIGHT}(e_1, e_2) \\ & \wedge \text{PAY-EVENT}(e_2) \wedge \text{PAYER}(e_2, \text{Asha}) \wedge \text{PAYEE}(e_2, \text{Boyang}) \end{aligned} \quad [13.3]$$

6470 As with first-order predicate calculus, the goal of event semantics is to provide a repre-
 6471 sentation that generalizes over many surface forms. Consider the following paraphrases
 6472 of (13.1):

- 6473 (13.4) Asha gives a book to Boyang.
- 6474 (13.5) A book is given to Boyang by Asha.
- 6475 (13.6) A book is given by Asha to Boyang.
- 6476 (13.7) The gift of a book from Asha to Boyang ...

6477 All have the same event semantic meaning, given in Equation 13.1. Note that the final
 6478 example does not include a verb! Events are often introduced by verbs, but not always:
 6479 in this final example, the noun *gift* introduces the same predicate, with the same accom-
 6480 panying arguments.

6481 **Semantic role labeling** (SRL) is a relaxed form of semantic parsing, in which each
 6482 semantic role is filled by a set of tokens from the text itself. This is sometimes called
 6483 “shallow semantics” because, unlike model-theoretic semantic parsing, role fillers need
 6484 not be symbolic expressions with denotations in some world model. A semantic role
 6485 labeling system is required to identify all predicates, and then specify the spans of text
 6486 that fill each role, when possible. To get a sense of the task, here is a more complicated
 6487 example:

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).

6488 (13.8) Boyang wants Asha to give him a linguistics book.

6489 In this example, there are two predicates, expressed by the verbs *want* and *give*. Thus, a
6490 semantic role labeler should return the following output:

- 6491 • (PREDICATE : *wants*, WANTED : *Boyang*, DESIRE : *Asha to give him a linguistics book*)
- 6492 • (PREDICATE : *give*, GIVER : *Asha*, RECIPIENT : *him*, GIFT : *a linguistics book*)

6493 In the example, *Boyang* and *him* may refer to the same person, but this would be ignored
6494 in semantic role labeling. However, in other predicate-argument representations, such
6495 as **Abstract Meaning Representation (AMR)**, such references must be resolved. We will
6496 return to AMR in § 13.3, but first, let us further consider the notion of semantic roles.

6497 13.1 Semantic roles

6498 As discussed so far, event semantics requires specifying a number of additional logical
6499 relations to link arguments to events: GIVER, RECIPIENT, SEER, SIGHT, etc. Indeed, every
6500 predicate requires a set of logical relations to express its own arguments. In contrast,
6501 adjuncts such as TIME and MANNER are shared across many types of events. A natural
6502 question is whether it is possible to treat mandatory arguments more like adjuncts, by
6503 identifying a set of generic argument types that are shared across many event predicates.
6504 This can be further motivated by examples involving semantically related verbs:

6505 (13.9) Asha gave Boyang a book.

6506 (13.10) Asha loaned Boyang a book.

6507 (13.11) Asha taught Boyang a lesson.

6508 (13.12) Asha gave Boyang a lesson.

6509 In the first two examples, the roles of Asha, Boyang, and the book are nearly identical. The
6510 third example is slightly different, but the fourth example shows that the roles of GIVER
6511 and TEACHER can be viewed as related.

6512 One way to think about the relationship between roles such as GIVER and TEACHER
6513 is by enumerating the set of properties that an entity typically possesses when it fulfills
6514 these roles: givers and teachers are usually animate and “volitional” (meaning that they
6515 choose to enter into the action).² In contrast, the thing that gets loaned or taught is usually
6516 not animate or volitional; furthermore, it is unchanged by the event.

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

6517 Building on these ideas, **thematic roles** generalize across predicates by leveraging the
 6518 shared semantic properties of typical role fillers (Fillmore, 1968). For example, in exam-
 6519 ples (13.9-13.12), Asha plays a similar role in all four sentences, which we will call the
 6520 **agent**. This reflects a number of shared semantic properties: she is the one who is actively
 6521 and intentionally performing the action, while Boyang is a more passive participant; the
 6522 book and the lesson would play a different role, as non-animate participants in the event.

6523 Let us now consider a few well-known approaches to semantic roles. Example anno-
 6524 tations from each of these systems are shown in Figure 13.1.

6525 13.1.1 VerbNet

6526 **VerbNet** (Kipper-Schuler, 2005) is a lexicon of verbs, and it includes thirty “core” thematic
 6527 roles played by arguments to these verbs. Here are some example roles, accompanied by
 6528 their definitions from the VerbNet Guidelines.³

- 6529 • AGENT: “ACTOR in an event who initiates and carries out the event intentionally or
 6530 consciously, and who exists independently of the event.”
- 6531 • PATIENT: “UNDERGOER in an event that experiences a change of state, location or
 6532 condition, that is causally involved or directly affected by other participants, and
 6533 exists independently of the event.”
- 6534 • RECIPIENT: “DESTINATION that is animate”
- 6535 • THEME: “UNDERGOER that is central to an event or state that does not have control
 6536 over the way the event occurs, is not structurally changed by the event, and/or is
 6537 characterized as being in a certain position or condition throughout the state.”

³http://verbs.colorado.edu/verb-index/VerbNet_Guidelines.pdf

- 6538 • TOPIC: “THEME characterized by information content transferred to another partic-
6539 ipant.”

6540 VerbNet roles are organized in a hierarchy, so that a TOPIC is a type of THEME, which in
6541 turn is a type of UNDERGOER, which is a type of PARTICIPANT, the top-level category.

6542 In addition, VerbNet organizes verb senses into a class hierarchy, in which verb senses
6543 that have similar meanings are grouped together. Recall from § 4.2 that multiple meanings
6544 of the same word are called **senses**, and that WordNet identifies senses for many English
6545 words. VerbNet builds on WordNet, so that verb classes are identified by the WordNet
6546 senses of the verbs that they contain. For example, the verb class *give*-13.1 includes
6547 the first WordNet sense of *loan* and the second WordNet sense of *lend*.

6548 Each VerbNet class or subclass takes a set of thematic roles. For example, *give*-13.1
6549 takes arguments with the thematic roles of AGENT, THEME, and RECIPIENT;⁴ the pred-
6550 icate TEACH takes arguments with the thematic roles AGENT, TOPIC, RECIPIENT, and
6551 SOURCE.⁵ So according to VerbNet, *Asha* and *Boyang* play the roles of AGENT and RECIP-
6552 IENT in the sentences,

6553 (13.13) Asha gave Boyang a book.

6554 (13.14) Asha taught Boyang algebra.

6555 The *book* and *algebra* are both THEMES, but *algebra* is a subcategory of THEME — a TOPIC
6556 — because it consists of information content that is given to the receiver.

6557 13.1.2 Proto-roles and PropBank

6558 Detailed thematic role inventories of the sort used in VerbNet are not universally accepted.
6559 For example, Dowty (1991, pp. 547) notes that “Linguists have often found it hard to agree
6560 on, and to motivate, the location of the boundary between role types.” He argues that a
6561 solid distinction can be identified between just two **proto-roles**, which have a number of
6562 distinguishing characteristics:

- 6563 • PROTO-AGENT: volitional involvement in the event or state; sentience and/or per-
6564 ception; causing an event or change of state in another participant; movement; exists
6565 independently of the event.

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

- 6566 • PROTO-PATIENT: undergoes change of state; causally affected by another participant;
 6567 stationary relative to the movement of another participant; does not exist
 6568 independently of the event.⁶

6569 In the examples in Figure 13.1, Asha has most of the proto-agent properties: in giving
 6570 the book to Boyang, she is acting volitionally (as opposed to *Boyang got a book from Asha*, in
 6571 which it is not clear whether Asha gave up the book willingly); she is sentient; she causes
 6572 a change of state in Boyang; she exists independently of the event. Boyang has some
 6573 proto-agent properties: he is sentient and exists independently of the event. But he also
 6574 some proto-patient properties: he is the one who is causally affected and who undergoes
 6575 change of state. The book that Asha gives Boyang has even fewer of the proto-agent
 6576 properties: it is not volitional or sentient, and it has no causal role. But it also lacks many
 6577 of the proto-patient properties: it does not undergo change of state, exists independently
 6578 of the event, and is not stationary.

6579 The **Proposition Bank**, or PropBank (Palmer et al., 2005), builds on this basic agent-
 6580 patient distinction, as a middle ground between generic thematic roles and predicate-
 6581 specific “deep roles.” Each verb is linked to a list of numbered arguments, with ARG0
 6582 as the proto-agent and ARG1 as the proto-patient. Additional numbered arguments are
 6583 verb-specific. For example, for the predicate TEACH,⁷ the arguments are:

- 6584 • ARG0: the teacher
 6585 • ARG1: the subject
 6586 • ARG2: the student(s)

6587 Verbs may have any number of arguments: for example, WANT and GET have five, while
 6588 EAT has only ARG0 and ARG1. In addition to the semantic arguments found in the frame
 6589 files, roughly a dozen general-purpose **adjuncts** may be used in combination with any
 6590 verb. These are shown in Table 13.1.

6591 PropBank-style semantic role labeling is annotated over the entire Penn Treebank. This
 6592 annotation includes the sense of each verbal predicate, as well as the argument spans.

6593 13.1.3 FrameNet

6594 Semantic **frames** are descriptions of situations or events. Frames may be **evoked** by one
 6595 of their **lexical units** (often a verb, but not always), and they include some number of

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.

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>know algebra</i> .
EXT	extent	<i>Prices increased</i> [AM-EXT <i>4%</i>].
CAU	cause	<i>Asha 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

6596 frame elements, which are like roles (Fillmore, 1976). For example, the act of teaching
 6597 is a frame, and can be evoked by the verb *taught*; the associated frame elements include
 6598 the teacher, the student(s), and the subject being taught. Frame semantics has played a
 6599 significant role in the history of artificial intelligence, in the work of Minsky (1974) and
 6600 Schank and Abelson (1977). In natural language processing, the theory of frame semantics
 6601 has been implemented in **FrameNet** (Fillmore and Baker, 2009), which consists of a lexicon
 6602 of roughly 1000 frames, and a corpus of more than 200,000 “exemplar sentences,” in which
 6603 the frames and their elements are annotated.⁸

6604 Rather than seeking to link semantic roles such as TEACHER and GIVER into the-
 6605 matic roles such as AGENT, FrameNet aggressively groups verbs into frames, and links
 6606 semantically-related roles across frames. For example, the following two sentences would
 6607 be annotated identically in FrameNet:

6608 (13.15) Asha taught Boyang algebra.

6609 (13.16) Boyang learned algebra from Asha.

6610 This is because *teach* and *learn* are both lexical units in the EDUCATION-TEACHING frame.
 6611 Furthermore, roles can be shared even when the frames are distinct, as in the following
 6612 two examples:

6613 (13.17) Asha gave Boyang a book.

6614 (13.18) Boyang got a book from Asha.

⁸These statistics are accurate at the time of this writing, in 2017. Current details can be found at the website, <https://framenet.icsi.berkeley.edu/>

6615 The GIVING and GETTING frames both have RECIPIENT and THEME elements, so Boyang
 6616 and the book would play the same role. Asha’s role is different: she is the DONOR in the
 6617 GIVING frame, and the SOURCE in the GETTING frame. FrameNet makes extensive use of
 6618 multiple inheritance to share information across frames and frame elements: for example,
 6619 the COMMERCE-SELL and LENDING frames inherit from GIVING frame.

6620 13.2 Semantic role labeling

6621 The task of semantic role labeling is to identify the parts of the sentence comprising the
 6622 semantic roles. In English, this task is typically performed on the PropBank corpus, with
 6623 the goal of producing outputs in the following form:

6624 (13.19) [ARG0 Asha] [GIVE.01 gave] [ARG2 Boyang’s mom] [ARG1 a book] [AM-TMP yesterday].

6625 Note that a single sentence may have multiple verbs, and therefore a given word may be
 6626 part of multiple role-fillers:

6627 (13.20) [ARG0 Asha] [WANT.01 wanted] [ARG1 Boyang to give her the book].

6628 Asha wanted [ARG0 Boyang] [GIVE.01 to give] [ARG2 her] [ARG1 the book].

6629

6630 13.2.1 Semantic role labeling as classification

6631 PropBank is annotated on the Penn Treebank, and annotators used phrasal constituents
 6632 (\S 9.2.2) to fill the roles. Therefore SRL can be viewed as the task of assigning to each
 6633 phrase a label from the set $\mathcal{R} = \{\emptyset, \text{PRED}, \text{ARG0}, \text{ARG1}, \text{ARG2}, \dots, \text{AM-LOC}, \text{AM-TMP}, \dots\}$,
 6634 where \emptyset indicates that the phrase plays no role, and PRED indicates that it is the verbal
 6635 predicate. If we treat semantic role labeling as a classification problem, we obtain the
 6636 following functional form:

$$\hat{y}_{(i,j)} = \underset{y}{\operatorname{argmax}} \theta \cdot f(\mathbf{w}, y, i, j, \rho, \tau), \quad [13.4]$$

6637 where,

6638 • (i, j) indicates the span of a phrasal constituent $(w_{i+1}, w_{i+2}, \dots, w_j)$;⁹

6639 • \mathbf{w} represents the sentence as a sequence of tokens;

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).

- 6640 • ρ is the index of the predicate verb in w ;
- 6641 • τ is the structure of the phrasal constituent parse of w .

6642 Table 13.2 shows the features used in the seminal paper on FrameNet semantic role
 6643 labeling by Gildea and Jurafsky (2002). By 2005 there were several systems for Prop-
 6644 Bank semantic role labeling, and their approaches and feature sets are summarized by
 6645 Carreras and Màrquez (2005). Typical features include: the phrase type, head word, part-
 6646 of-speech, boundaries, and neighbors of the proposed argument $w_{i:j}$; the word, lemma,
 6647 part-of-speech, and voice of the verb w_ρ (active or passive), as well as features relating
 6648 to its frameset; the distance and path between the verb and the proposed argument. In
 6649 this way, semantic role labeling systems are high-level “consumers” in the NLP stack, us-
 6650 ing features produced from lower-level components such as part-of-speech taggers and
 6651 parsers. More comprehensive feature sets are enumerated by Das et al. (2014) and Täck-
 6652 ström et al. (2015).

6653 A particularly powerful class of features relate to the **syntactic path** between the ar-
 6654 gument and the predicate. These features capture the sequence of moves required to get
 6655 from the argument to the verb by traversing the phrasal constituent parse of the sentence.
 6656 The idea of these features is to capture syntactic regularities in how various arguments
 6657 are realized. Syntactic path features are best illustrated by example, using the parse tree

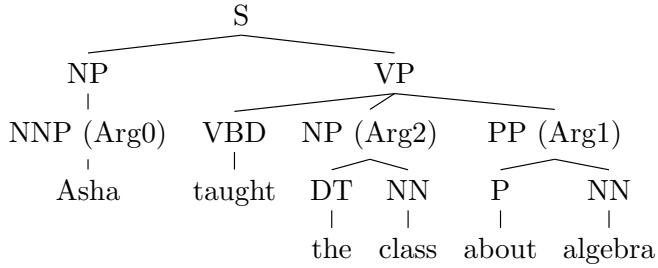


Figure 13.2: Semantic role labeling is often performed on the parse tree for a sentence, labeling individual constituents. [todo: check arg1; show arrows for path features]

6658 in Figure 13.2:

- 6659 • The path from *Asha* to the verb *taught* is $\text{NNP} \uparrow \text{NP} \uparrow \text{S} \downarrow \text{VP} \downarrow \text{VBD}$. The first part of
 6660 the path, $\text{NNP} \uparrow \text{NP} \uparrow \text{S}$, means that we must travel up the parse tree from the NNP
 6661 tag (proper noun) to the S (sentence) constituent. The second part of the path,
 6662 $\text{S} \downarrow \text{VP} \downarrow \text{VBD}$, means that we reach the verb by producing a VP (verb phrase) from
 6663 the S constituent, and then by producing a VBD (past tense verb). This feature is
 6664 consistent with *Asha* being in subject position, since the path includes the sentence
 6665 root S.
- 6666 • The path from *the class* to the verb is $\text{NP} \uparrow \text{VP} \downarrow \text{VBD}$. This is consistent with *the class*
 6667 being in object position, since the path passes through the VP node that dominates
 6668 the verb *taught*.

6669 Because there are many possible path features, it can also be helpful to look at smaller
 6670 parts: for example, the upward and downward parts can be treated as separate features;
 6671 another feature might consider whether S appears anywhere in the path.

6672 Rather than using the constituent parse, it is also possible to build features from the
 6673 **dependency path** between the head word of each argument and the verb (Pradhan et al.,
 6674 2005). Using the Universal Dependency part-of-speech tagset and dependency relations (Nivre
 6675 et al., 2016), the dependency path from *Asha* to *taught* is $\text{PROPN} \xleftarrow[\text{NSUBJ}]{} \text{VERB}$, because *taught*
 6676 is the head of a relation of type $\xleftarrow[\text{NSUBJ}]{} \text{VERB}$ with *Asha*. Similarly, the dependency path from *class*
 6677 to *taught* is $\text{NOUN} \xleftarrow[\text{DOBJ}]{} \text{VERB}$, because *class* heads the noun phrase that is a direct object of
 6678 *taught*. A more interesting example is *Asha tried to teach the class*, where the path from
 6679 *Asha* to *teach* is $\text{PROPN} \xleftarrow[\text{NSUBJ}]{} \text{VERB} \rightarrow[\text{XCOMP}] \text{VERB}$. The right-facing arrow in second relation
 6680 indicates that *tried* is the head of its XCOMP relation with *teach*.

6681 **13.2.2 Semantic role labeling as constrained optimization**

6682 A potential problem with treating SRL as a classification problem is that there are a num-
 6683 ber of sentence-level **constraints**, which a classifier might violate.

- 6684 • For a given verb, there can be only one argument of each type (ARG0, ARG1, etc.)
 6685 • Arguments cannot overlap. This problem arises when we are labeling the phrases
 6686 in a constituent parse tree, as shown in Figure 13.2: if we label the PP *about algebra*
 6687 as an argument or adjunct, then its children *about* and *algebra* must be labeled as \emptyset .
 6688 The same constraint also applies to the syntactic ancestors of this phrase.

6689 These constraints can be viewed as introducing dependencies across labeling deci-
 6690 sions. In structure prediction problems such as sequence labeling and parsing, such de-
 6691 pendencies are usually handled by defining additional features over the entire structure,
 6692 y . Efficient inference requires that the global features have a local decomposition that
 6693 enables dynamic programming: for example, in sequence labeling, the features over y
 6694 were decomposed into features over pairs of adjacent tags, permitting the application of
 6695 the Viterbi algorithm for inference. Unfortunately, the constraints that arise in semantic
 6696 role labeling are less amenable to local decomposition — particularly the constraint that
 6697 each argument is used only once in the sentence.¹⁰ We therefore consider **constrained**
 6698 **optimization** as an alternative solution.

Let the **feasible set** $\mathcal{C}(\tau)$ refer to all labelings that obey the constraints introduced by the parse τ . We can reformulate the semantic role labeling problem as a constrained optimization,

$$\begin{aligned} \max_{\mathbf{y}} \quad & \sum_{(i,j) \in \tau} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, y_{(i,j)}, i, j, \rho, \tau) \\ \text{s.t.} \quad & \mathbf{y} \in \mathcal{C}(\tau). \end{aligned} \quad [13.5]$$

6699 In this formulation, the objective (shown on the first line) is a separable function of each
 6700 individual labeling decision, but the constraints (shown on the second line) apply to the
 6701 overall labeling. The sum $\sum_{(i,j) \in \tau}$ indicates that we are summing over all constituent
 6702 spans in the parse τ . The expression *s.t.* in the second line means that we maximize the
 6703 objective *subject to* the constraints that appear to the right.

6704 **13.2.2.1 Integer linear programming**

6705 A number of practical algorithms exist for restricted forms of constrained optimization.
 6706 One such restricted form is **integer linear programming**, in which we optimize a linear

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.

6707 objective function over integer variables, with linear constraints. To formulate SRL as
 6708 an integer linear program, we begin by rewriting the labels as a set of binary variables
 6709 $\mathbf{z} = \{z_{i,j,r}\}$, where,

$$z_{i,j,r} = \begin{cases} 1, & y_{(i,j)} = r \\ 0, & \text{otherwise.} \end{cases} \quad [13.6]$$

6710 Thus, the variables \mathbf{z} are a binarized version of the semantic role labeling \mathbf{y} .

6711 **Objective** Next, we restrict the objective to be a linear function of \mathbf{z} . We begin with
 6712 the feature function, $f_j(\mathbf{w}, y_{(i,j)}, i, j, \rho, \tau)$. Such features are typically logical conjunctions
 6713 involving the label $y_{(i,j)}$. For example:

$$f_j(\mathbf{w}, y_{(i,j)}, i, j, \rho, \tau) = \begin{cases} 1, & y_{(i,j)} = \text{ARG1} \wedge w_i = \text{the} \\ 0, & \text{otherwise.} \end{cases} \quad [13.7]$$

This feature is an indicator that takes the value 1 for constituents that are labeled ARG1 and begin with the word *the*. If all features are conjunctions with the label, then the feature function can be rewritten,

$$\mathbf{f}(\mathbf{w}, y_{(i,j)}, i, j, \rho, \tau) = \sum_{r \in \mathcal{R}} z_{i,j,r} \times \mathbf{g}(\mathbf{w}, i, j, r, \rho, \tau), \quad [13.8]$$

where \mathcal{R} is the set of all possible labels, $\{\text{ARG0}, \text{ARG1}, \dots, \text{AM-LOC}, \dots, \emptyset\}$. With this change in notation, we can now rewrite the objective,

$$\sum_{(i,j) \in \tau} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, y_{(i,j)}, i, j, \rho, \tau) = \sum_{(i,j) \in \tau} \boldsymbol{\theta} \cdot \left(\sum_{r \in \mathcal{R}} z_{i,j,r} \times \mathbf{g}(\mathbf{w}, i, j, r, \rho, \tau) \right) \quad [13.9]$$

$$= \sum_{(i,j) \in \tau} \sum_{r \in \mathcal{R}} z_{i,j,r} (\boldsymbol{\theta} \cdot \mathbf{g}(\mathbf{w}, i, j, r, \rho, \tau)) \quad [13.10]$$

$$= \sum_{(i,j) \in \tau} \sum_{r \in \mathcal{R}} z_{i,j,r} \psi_{i,j,r}, \quad [13.11]$$

6714 where $\psi_{i,j,r} \triangleq \boldsymbol{\theta} \cdot \mathbf{g}(\mathbf{w}, i, j, r, \rho, \tau)$. This objective is clearly a linear function of the variables
 6715 $\mathbf{z} = \{z_{i,j,r}\}$.

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,

$$\forall r \neq \emptyset, \quad \sum_{(i,j) \in \tau} z_{i,j,r} \leq 1. \quad [13.12]$$

6716 Recall that $z_{i,j,r} = 1$ if and only if the span (i, j) has label r ; this constraint says that for
 6717 each possible label $r \neq \emptyset$, there can be at most one (i, j) such that $z_{i,j,r} = 1$. This constraint
 6718 can be written in the form $\mathbf{A}z \leq b$, as you will find if you complete the exercises at the
 6719 end of the chapter.

Now consider the constraint that labels cannot overlap. Let the function $\pi_\tau(i, j) = \{(i', j')\}$ indicate the set of constituents that are ancestors or descendants of (i, j) in the parse τ . For any (i, j) such that $y_{i,j} \neq \emptyset$, the non-overlapping constraint means that all of its ancestors and descendants (i', j') must be labeled as a non-argument, $y_{i',j'} = \emptyset$. We can write this as a set of linear constraints,

$$\forall (i, j) \in \tau, \quad \sum_{r \neq \emptyset} \left(z_{i,j,r} + \sum_{(i', j') \in \pi_\tau(i, j)} z_{i',j',r} \right) \leq 1. \quad [13.13]$$

We can therefore rewrite the semantic role labeling problem as the following integer linear program,

$$\max_{z \in \{0,1\}^{|\tau|}} \quad \sum_{(i,j) \in \tau} \sum_{r \in \mathcal{R}} z_{i,j,r} \psi_{i,j,r} \quad [13.14]$$

$$s.t. \quad \forall r \neq \emptyset, \quad \sum_{(i,j) \in \tau} z_{i,j,r} \leq 1. \quad [13.15]$$

$$\forall (i, j) \in \tau, \quad \sum_{r \neq \emptyset} \left(z_{i,j,r} + \sum_{(i', j') \in \pi_\tau(i, j)} z_{i',j',r} \right) \leq 1. \quad [13.16]$$

6720 The effectiveness of integer linear programming for semantic role labeling was first
 6721 demonstrated by Punyakanok et al. (2008).

6722 **Learning with constraints** Learning can be performed in the context of constrained op-
 6723 timization using the usual perceptron or large-margin classification updates. Because
 6724 constrained inference is generally more time-consuming, a key question is whether it is
 6725 necessary to apply the constraints during learning. Chang et al. (2008) find that better per-
 6726 formance can be obtained by learning *without* constraints, and then applying constraints
 6727 only when using the trained model to predict semantic roles for unseen data.

6728 **How important are the constraints?** Das et al. (2014) find that an unconstrained, classification-
 6729 based method performs nearly as well as constrained optimization for FrameNet parsing;
 6730 while it commits many violations of the “no-overlap” constraint, the overall F_1 score is
 6731 less than one point worse than the score at the constrained optimum. Similar results
 6732 are obtained for PropBank semantic role labeling by Punyakanok et al. (2008). He et al.

(2017) find that constrained inference makes a bigger impact if the constraints are based on manually-labeled “gold” syntactic parses. This implies that errors from the syntactic parser may limit the effectiveness of the constraints. Punyakanok et al. (2008) hedge against parser error by including constituents from several different parsers; any constituent can be selected from any parse, and additional constraints ensure that overlapping constituents are not selected.

Implementation Integer linear programming solvers such as `glpk`,¹¹ `cplex`,¹² and `Gurobi`¹³ allow inequality constraints to be expressed directly in the problem definition, rather than in the matrix form $Az \leq b$. The time complexity of integer linear programming is theoretically exponential in the number of variables $|z|$, but in practice these off-the-shelf solvers obtain good solutions efficiently. Das et al. (2014) report that the `cplex` solver requires 43 seconds to perform inference on the FrameNet test set, which contains 4,458 predicates.

Recent work has shown that many constrained optimization problems in natural language processing can be solved in a highly parallelized fashion, using optimization techniques such as **dual decomposition**, which are capable of exploiting the underlying problem structure (Rush et al., 2010). Das et al. (2014) apply this technique to FrameNet semantic role labeling, obtaining an order-of-magnitude speedup over `cplex`.

13.2.3 Neural semantic role labeling

Neural network approaches to SRL have tended to treat it as a sequence labeling task, using a labeling scheme such as the **BIO notation**, which we previously saw in named entity recognition (§ 8.3). In this notation, the first token in a span of type ARG1 is labeled B-ARG1; all remaining tokens in the span are **inside**, and are therefore labeled I-ARG1. Tokens outside any argument are labeled O. For example:

(13.21) *Asha taught Boyang 's mom about algebra*
 B-ARG0 PRED B-ARG2 I-ARG2 I-ARG2 B-ARG1 I-ARG1

We now consider two classes of neural networks that can learn to produce such labelings.

Convolutional neural networks One of the first applications of **convolutional neural networks** (§ 3.4) to natural language processing was the task of classifying semantic roles. Collobert et al. (2011) treat the task as a classification problem, using information gathered from across the sentence to compute the label for each token. For example, suppose our goal is to tag the role of word m with respect to verb v ; then for word n , we compute the discrete feature vector, $f(w, n, v, m)$, which would include the lower-case word w_m , and

¹¹<https://www.gnu.org/software/glpk/>

¹²<https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>

¹³<http://www.gurobi.com/>

the distances $m - n$ and $m - v$. These features are then used as the inputs to a nonlinear prediction model, as follows:

- Each discrete feature is associated with a dense vector embedding, and these embeddings are concatenated, resulting in a dense vector $\mathbf{x}_m^{(0)}$. The horizontal concatenation of the dense embeddings for all words in the sentence is $\mathbf{X}^{(0)} = [\mathbf{x}_0^{(0)}, \mathbf{x}_1^{(0)}, \dots, \mathbf{x}_M^{(0)}]$.
- Next, a convolutional operation is applied to merge information across words, $\mathbf{X}^{(1)} = \mathbf{C}\mathbf{X}^{(0)}$. Thus, $\mathbf{x}_m^{(1)}$ contains information about the word w_m , but also about its near neighbors. (Special padding vectors are included on the left and right ends of the matrix $\mathbf{X}^{(0)}$ before convolution.)
- To convert the matrix $\mathbf{X}^{(1)}$ back to a vector $\mathbf{z}^{(1)}$, Collobert *et al.* apply **max pooling**. We will write $\mathbf{z} = \text{MaxPool}(\mathbf{X})$ to indicate that each $z_j = \max_m(x_{0,j}^{(1)}, x_{1,j}^{(1)}, \dots, x_{M,j}^{(1)})$.
- The vector $\mathbf{z}^{(1)}$ is then passed through several feedforward layers, $\mathbf{z}^{(i)} = \mathbf{g}(\Theta^{(i)}\mathbf{z}^{(i-1)})$, where $\Theta^{(i)}$ is a matrix of weights and \mathbf{g} is an elementwise nonlinear transformation.¹⁴
- At the output layer $\mathbf{z}^{(K)}$ is used to make a prediction, $\hat{y} = \text{argmax}_y \Theta^{(y)}\mathbf{z}^{(K)}$. [todo: consider making this a figure/algorithm]

This model is very similar to the classifier described in Figure 3.4; the key difference is that the inclusion of distance feature embeddings to compute distinct labels for each word in the span. Collobert et al. (2011) apply this model without regard to constraints, so in principle it could produce labelings that include each argument multiple times. The parameters of the model include the word and feature embeddings that constitute $\mathbf{X}^{(0)}$, the convolution matrix \mathbf{C} , the feedforward weight matrices $\Theta^{(i)}$, and the prediction weights $\Theta^{(y)}$. Each of these parameters is estimated by backpropagated stochastic gradient descent (see § 6.3.1). Collobert et al. (2011) emphasize that **multi-task learning** was essential to get good performance: they train the word embeddings not only to accurately predict PropBank labels, but also to assign a high likelihood to a large corpus of unlabeled data. A more contemporary approach would be to use **pre-trained word embeddings**, which have already been trained to predict words in context, thereby avoiding the cost of jointly training across a large unlabeled dataset.

¹⁴Collobert *et al.* use a piecewise linear **hard tanh** function for nonlinear transformations,

$$g(x) = \begin{cases} -1, & x < -1 \\ x, & -1 \leq x \leq 1 \\ 1, & x > 1. \end{cases} \quad [13.17]$$

Like the **rectified linear unit (ReLU)** (§ 3.2.1), this function is piecewise linear, and has a gradient that is easy to compute.

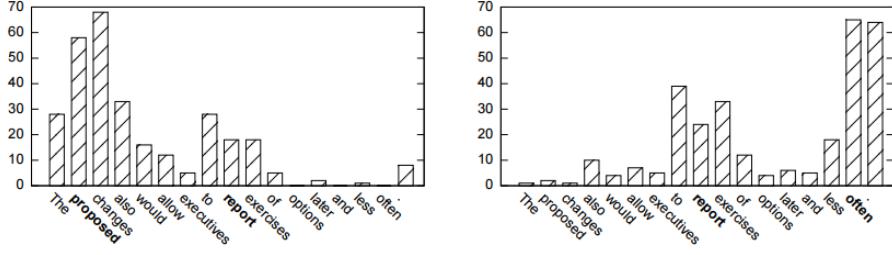


Figure 13.3: Number of features chosen at each word position by the max pooling operation, for tagging the words *proposed* (left) and *often* (right). Figure reprinted from Collobert et al. (2011) [todo: permission]

6793 A key aspect of convolutional neural networks is the use of **pooling** operations, which
 6794 combine information across a variable-length sequence of vectors into a single vector or
 6795 matrix. Max pooling is widely used in natural language processing applications, because
 6796 it enables each element in the vector \mathbf{z} to take information from across a sentence or other
 6797 sequence of text. Figure 13.3 shows the number of “features” taken from each word in
 6798 a sentence — that is, how often the max operation chooses an element from each word.
 6799 In each case, the pooling operation emphasizes the word to be tagged, its neighbors, and
 6800 also the main verb *report*.

Recurrent neural networks An alternative neural approach to semantic role labeling is to use **recurrent neural network** models, such as **long short-term memories** (LSTMs; see § 7.6 to review how these models are applied to tagging tasks). Zhou and Xu (2015) apply a bidirectional multilayer LSTM 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,

$$\mathbf{Z}^{(1)} = \text{BiLSTM}(\mathbf{X}) \quad [13.18]$$

$$\mathbf{Z}^{(i)} = \text{BiLSTM}(\mathbf{Z}^{(i-1)}) \quad [13.19]$$

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}} \sum_{m=1}^M \Theta^{(y)} \mathbf{z}_m^{(K)} + \psi_{y_{m-1}, y_m}. \quad [13.20]$$

6801 Note that the final step maximizes over the entire labeling \mathbf{y} , and includes a score for
 6802 each tag transition ψ_{y_{m-1}, y_m} . This combination of LSTM and pairwise potentials on tags
 6803 is an example of an **LSTM-CRF**. The maximization over \mathbf{y} is performed by the Viterbi
 6804 algorithm.

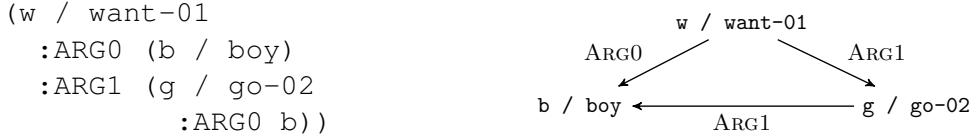


Figure 13.4: Two views of the AMR representation for the sentence *The boy wants to go.*

6805 This model strongly outperformed alternative approaches at the time, including con-
 6806 strained decoding and convolutional neural networks. More recent work has combined
 6807 recurrent neural network models with constrained decoding, using the A^* search algo-
 6808 rithm to search over labelings that are feasible with respect to the constraints (He et al.,
 6809 2017). This yields small improvements over the method of Zhou and Xu (2015). He et al.
 6810 (2017) obtain larger improvements by creating an **ensemble** of SRL systems, each trained
 6811 on an 80% subsample of the corpus. The average prediction across this ensemble is more
 6812 robust than any individual model.

6813 13.3 Abstract Meaning Representation

6814 Semantic role labeling transforms the task of semantic parsing to a labeling task. Consider
 6815 the sentence,

6816 (13.22) The boy wants to go.

6817 The PropBank semantic role labeling analysis is:

- 6818 • (PREDICATE : *wants*, ARG0 : *the boy*, ARG1 : *to go*)
- 6819 • (PREDICATE : *go*, ARG1 : *the boy*)

6820 The **Abstract Meaning Representation (AMR)** unifies this analysis into a graph struc-
 6821 ture, in which each node is a **variable**, and each edge indicates a **concept** (Banarescu
 6822 et al., 2013). This can be written in two ways, as shown in Figure 13.4. On the left is the
 6823 PENMAN notation (Matthiessen and Bateman, 1991), in which each set of parentheses
 6824 introduces a variable. Each variable is an **instance** of a concept, which is indicated with
 6825 the slash notation: for example, *w / want-01* indicates that the variable *w* is an instance
 6826 of the concept *want-01*, which in turn refers to the PropBank frame for the first sense
 6827 of the verb *want*. Relations are introduced with colons: for example, *:arg0 (b / boy)*
 6828 indicates a relation of type *arg0* with the newly-introduced variable *b*. Variables can be
 6829 reused, so that when the variable *b* appears again as an argument to *g*, it is understood to
 6830 refer to the same boy in both cases. This arrangement is indicated compactly in the graph
 6831 structure on the right, with edges indicating concepts.

6832 AMR differs from PropBank-style semantic role labeling in a few key ways. First, it
 6833 reifies each entity as a variable: for example, the *boy* in (13.22) is reified in the variable
 6834 *b*, which is reused as ARG0 in its relationship with *w* / want-01, and as ARG1 in its
 6835 relationship with *g* / go-02. Reifying entities as variables also makes it possible to
 6836 represent the substructure of noun phrases more explicitly. For example, *Asha borrowed*
 6837 *the algebra book* would be represented as:

```
6838 (b / borrow-01
6839   :ARG0 (p / person
6840     :name (n / name
6841       :op1 "Asha"))
6842   :ARG1 (b2 / book
6843     :topic (a / algebra))
```

6844 This indicates that the variable *p* is a person, whose *name* is the variable *n*; that *n*
 6845 has one token, the string *Asha*. Similarly, the variable *b2* is a book, and the *topic* of *b2*
 6846 is a variable *a* whose type is *algebra*. The relations *name* and *topic* are examples of
 6847 **non-core roles**, which are similar to adjunct modifiers in PropBank. However, AMR's
 6848 inventory is more extensive, including more than 70 non-core roles, such as negation,
 6849 time, manner, frequency, and location. Lists and sequences — such as the list of tokens in
 6850 a name — are described using the roles *op1*, *op2*, etc.

6851 Another key feature of AMR is that a semantic predicate can be introduced by any
 6852 syntactic element. Consider the following examples, from Banerescu et al. (2013):

- 6853 (13.23) The boy destroyed the room.
- 6854 (13.24) the destruction of the room by the boy ...
- 6855 (13.25) the boy's destruction of the room ...

6856 All these examples have the same semantics in AMR,

```
6857 (d / destroy-01
6858   :arg0 (b / boy)
6859   :arg1 (r / room))
```

6860 Note that the noun *destruction* is linked to the verb *destroy*, which is captured by the Prop-
 6861 Bank frame *destroy-01*. This can happen with adjectives as well: in the phrase *the*
 6862 *attractive spy*, the adjective *attractive* is linked to the PropBank frame *attract-01*:

```
6863 (s / spy
6864   :arg0-of (a / attract-01))
```

6865 In this example, `arg0-of` is an **inverse relation**, indicating that `s` is the `arg0` of the
6866 predicate `a`. Inverse relations make it possible for all AMR parses to have a single root
6867 concept, which should be the **focus** of the utterance.

6868 There are a number of other important linguistic issues in the design of AMR, which
6869 are summarized in the original paper (Banarescu et al., 2013) and the tutorial slides by
6870 Schneider et al. (2015). While AMR goes farther than semantic role labeling, it does not
6871 link semantically-related frames such as `buy/sell` (as FrameNet does), does not han-
6872 dle quantification (as first-order predicate calculus does), and makes no attempt to han-
6873 dle noun number and verb tense (as PropBank does). A recent survey by Abend and
6874 Rappoport (2017) situates AMR with respect to several other semantic representation
6875 schemes.

6876 13.3.1 AMR Parsing

6877 Abstract Meaning Representation is not a labeling of the original text — unlike PropBank
6878 semantic role labeling, and most of the other tagging and parsing tasks that we have
6879 encountered thus far. The AMR for a given sentence may include multiple concepts for
6880 single words in the sentence: as we have seen, the sentence *Asha likes algebra* contains both
6881 person and name concepts for the word *Asha*. Conversely, words in the sentence may not
6882 appear in the AMR: in *Boyang made a tour of campus*, the **light verb** `make` would not appear
6883 in the AMR, which would instead be rooted on the predicate `tour`. As a result, AMR
6884 is difficult to parse, and even evaluating AMR parsing involves considerable algorithmic
6885 complexity (Cai and Yates, 2013).

6886 A further complexity is that AMR labeled datasets do not explicitly show the **align-
6887 ment** between the AMR annotation and the words in the sentence. For example, the link
6888 between the word *wants* and the concept `want-01` is not annotated. To acquire train-
6889 ing data for learning-based parsers, it is therefore necessary to first perform an alignment
6890 between the training sentences and their AMR parses. Flanigan et al. (2014) introduce a
6891 rule-based parser, which links text to concepts through a series of increasingly high-recall
6892 steps.

6893 **Graph-based parsing** One family of approaches to AMR parsing is similar to the graph-
6894 based methods that we encountered in syntactic dependency parsing (chapter 11). For
6895 these systems (Flanigan et al., 2014), parsing is a two-step process:

- 6896 1. **Concept identification** (Figure 13.5a). This involves constructing concept subgraphs
6897 for individual words or spans of adjacent words. For example, in the sentence,
6898 *Asha likes algebra*, we would hope to identify the minimal subtree including just the
6899 concept `like-01` for the word *like*, and the subtree (`p / person :name (n /`
6900 `name :op1 Asha)`) for the word *Asha*.

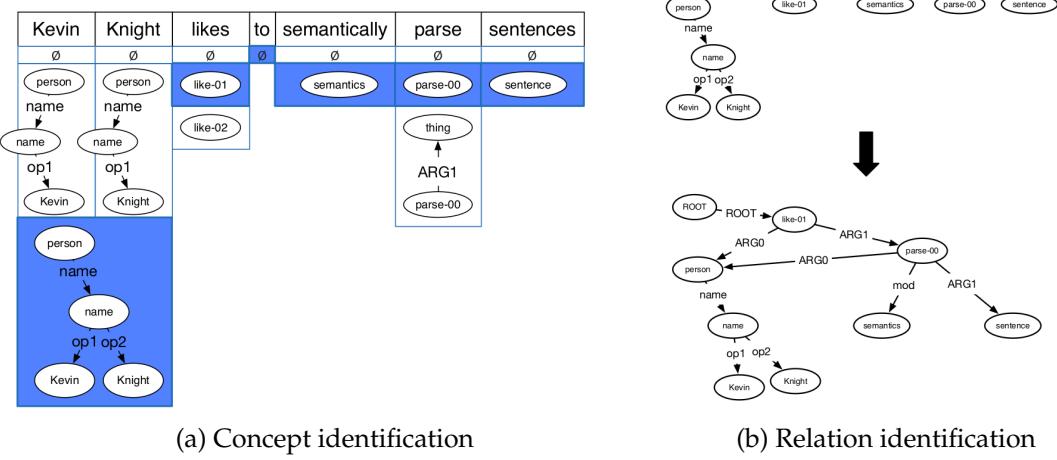


Figure 13.5: Subtasks for Abstract Meaning Representation parsing, from Schneider et al. (2015). [todo: ask for permission, or remake]

6901 2. **Relation identification** (Figure 13.5b). This involves building a directed graph over
 6902 the concepts, where the edges are labeled by the relation type. AMR imposes a
 6903 number of constraints on the graph: all concepts must be included, the graph must
 6904 be **connected** (there must be a path between every pair of nodes in the undirected
 6905 version of the graph), and every node must have at most one outgoing edge of each
 6906 type.

6907 Both of these problems are solved by structure prediction. Concept identification re-
 6908 quires simultaneously segmenting the text into spans, and labeling each span with a graph
 6909 fragment containing one or more concepts. This is done by computing a set of features
 6910 for each candidate span s and concept labeling c , and then returning the labeling with the
 6911 highest overall score.

6912 Relation identification can be formulated as search for the maximum spanning sub-
 6913 graph, under a set of constraints. Each labeled edge has a score, which is computed from
 6914 features of the concepts. We then search for the set of labeled edges that maximizes the
 6915 sum of these scores, under the constraint that the resulting graph is well-formed AMR.
 6916 § 13.2.2 described how constrained optimization can be used for semantic role labeling;
 6917 similar techniques have been applied to AMR relation identification (Flanigan et al., 2014).

6918 **Transition-based parsing** In many cases, AMR parses are structurally similar to syntac-
 6919 tic dependency parses. Figure 13.6 shows one such example. This motivates an alternative
 6920 approach to AMR parsing: simply modify the syntactic dependency parse until it looks
 6921 like a good AMR parse. Wang et al. (2015) propose a transition-based method, based on

(c) Jacob Eisenstein 2018. Work in progress.

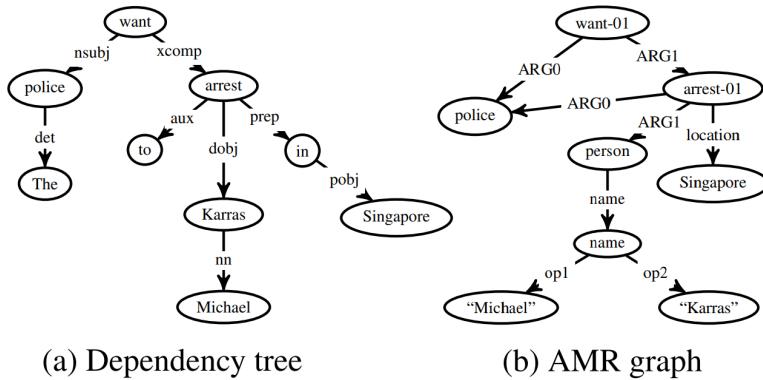


Figure 13.6: Syntactic dependency parse and AMR graph for the sentence *The police want to arrest Michael Karras in Singapore* (borrowed from Wang et al. (2015)) [todo: permission]

incremental modifications to the syntactic dependency tree (you may review transition-based dependency parsing in § 11.3). At each step, the parser performs an action: for example, adding an AMR relation label to the current dependency edge, swapping the direction of a syntactic dependency edge, or cutting an edge and reattaching the orphaned subtree to a new parent. They train their system as a classifier, learning to choose the action as would be given by an **oracle** that is capable of reproducing the ground-truth parse. The 2016 SemEval evaluation compared a number of contemporary AMR parsing systems (May, 2016).

6930 13.4 Applications of Predicate-Argument Semantics

Question answering Factoid questions have answers that are single words or phrases, such as *who discovered priors?*, *where was Barack Obama born?*, and *in what year did the Knicks last win the championship?* Shen and Lapata (2007) show that semantic role labeling can be used to answer such questions, by linking them to sentences in a corpus of text. They perform FrameNet semantic role labeling, making heavy use of dependency path features. For each sentence, they produce a weighted bipartite graph¹⁵ between FrameNet semantic roles and the words and phrases in the sentence. This is done by first scoring all pairs of semantic roles and assignments, as shown in the top half of Figure 13.8. They then find the bipartite edge cover, which is the minimum weighted subset of edges such that each vertex has at least one edge, as shown in the bottom half of Figure 13.8. After analyzing the question in this manner, Shen and Lapata then find semantically-compatible sentences in the corpus, by performing graph matching on the bipartite graphs for the question and

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.

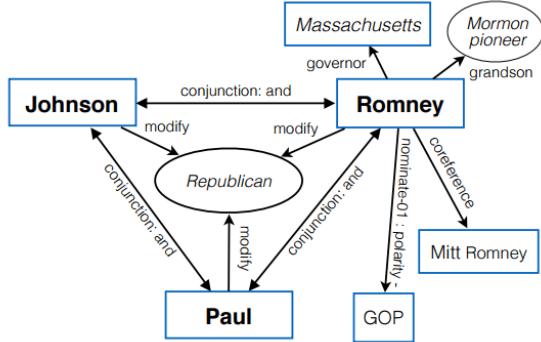


Figure 13.7: Fragment of AMR knowledge network for entity linking. Figure reprinted from Pan et al. (2015) [todo: permission]

6943 candidate answer sentences. Finally, the **expected answer phrase** in the question — typi-
 6944 cally the *wh*-word — is linked to a phrase in the candidate answer source, and that phrase
 6945 is returned as the answer.

6946 **Relation extraction** The task of **relation extraction** involves identifying pairs of entities
 6947 for which a given semantic relation holds. For example, we might like to find all (i, j)
 6948 such that i is the INVENTOR-OF j . PropBank semantic role labeling can be applied to this
 6949 task by identifying sentences whose verb signals the desired relation, and then extracting
 6950 ARG1 and ARG2 as arguments. (To fully solve this task, these arguments must then be
 6951 linked to entities, as described in chapter 17.) Christensen et al. (2010) compare the UIUC
 6952 semantic role labeling system (which uses integer linear programming) against a simpler
 6953 approach based on surface patterns (Banko et al., 2007). They find that the SRL system is
 6954 considerably more accurate, but that it is several orders of magnitude slower. Conversely,
 6955 Barnickel et al. (2009) apply SENNA, a convolutional neural network SRL system (Col-
 6956 lobert and Weston, 2008) to the task of identifying biomedical relations (e.g., which genes
 6957 inhibit or activate each other). In comparison with a strong baseline that applies a set of
 6958 rules to syntactic dependency structures (Fundel et al., 2007), the SRL system is faster but
 6959 less accurate. One possible explanation for these divergent results is that the Fundel et al.
 6960 compare against a baseline which is carefully tuned for performance in a relatively nar-
 6961 row domain, while the system of Banko et al. is designed to analyze text across the entire
 6962 web.

6963 **Entity linking** Another core task in information extraction is to link mentions of enti-
 6964 ties (e.g., *Republican candidates like Romney, Paul, and Johnson ...*) to entities in a knowl-
 6965 edge base (e.g., Lyndon Johnson or Gary Johnson). This is often done by examining
 6966 nearby “collaborator” mentions — in this case, *Romney* and *Paul*. By jointly linking all

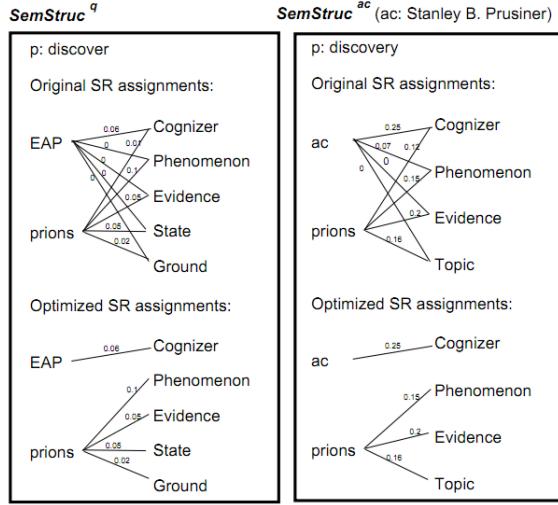


Figure 13.8: FrameNet semantic role labeling is used in factoid question answering, by aligning the semantic roles in the question (q) against those of sentences containing answer candidates (ac). “EAP” is the expected answer phrase, replacing the word *who* in the question. Figure reprinted from Shen and Lapata (2007) [todo: permission]

such mentions, it is possible to arrive at a good overall solution. Pan et al. (2015) apply AMR to this problem. For each entity, they construct a knowledge network based on its semantic relations with other mentions within the same sentence. They then rerank a set of candidate entities, based on the overlap between the entity’s knowledge network and the semantic relations present in the sentence (Figure 13.7). When applied to manually labeled AMR annotations, this approach is superior to state-of-the-art supervised methods that have access to labeled examples of linked mentions. Pan et al. also show that the method performs well from automated AMR, and that an AMR-based approach far outperforms a similar method based on PropBank semantic role labeling.[todo: rework for clarity]

Exercises

1. Write out an event semantic representation for the following sentences. You may make up your own predicates.
 - (13.26) *Abigail shares with Max.*
 - (13.27) *Abigail reluctantly shares a toy with Max.*
 - (13.28) *Abigail hates to share with Max.*

- 6983 2. Find the PropBank framesets for *share* and *hate* at <http://verbs.colorado.edu/propbank/framesets-english-aliases/>, and rewrite your answers from the
6984 6985 previous question, using the thematic roles ARG0, ARG1, and ARG2.
- 6986 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
6987 6988 sure about the parse, you can try an online parser such as <http://nlp.stanford.edu:8080/parser/>.
6989
- 6990 4. Compute the dependency path features for Abigail and Max in each of the example
6991 6992 sentences (13.26) and (13.28) in Question 1, with respect to the verb *share*. Again, if
6993 6994 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).
- 6995 5. PropBank semantic role labeling includes **reference arguments**, such as,

6996 (13.29) [AM-LOC The bed] on [R-AM-LOC which] I slept broke.¹⁶

6997 The label R-AM-LOC indicates that word *which* is a reference to *The bed*, which ex-
6998 6999 presses the location of the event. Reference arguments must have referents: the tag
7000 7001 R-AM-LOC can appear only when AM-LOC also appears in the sentence. Show how
7002 7003 to express this as a linear constraint, specifically for the tag R-AM-LOC. Be sure to
7004 7005 correctly handle the case in which neither AM-LOC nor R-AM-LOC appear in the
7006 7007 sentence.

- 7003 6. Explain how to express the constraints on semantic role labeling in Equation 13.12
7004 7005 and Equation 13.13 in the general form $Az \geq b$.
- 7006 7. Download the FrameNet sample data (<https://framenet.icsi.berkeley.edu/fndrupal/fulltextIndex>), and train a bag-of-words classifier to predict the
7007 7008 frame that is evoked by each verb in each example. Your classifier should build
7009 7010 a bag-of-words from the sentence in which the frame-evoking lexical unit appears.
7011 7012 [todo: Somehow limit to one or a few lexical units.] [todo: use NLTK if possible]
- 7013 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
7014 7015 LSTM to predict tags. You will have to convert the downloaded instances to a BIO
7016 7017 sequence labeling representation first.
- 7018 9. Produce the AMR annotations for the following examples:

¹⁶Example from 2013 NAACL tutorial slides by Shumin Wu

- 7015 (13.30) The girl likes the boy.
 7016 (13.31) The girl was liked by the boy.
 7017 (13.32) Abigail likes Maxwell Aristotle.
 7018 (13.33) The spy likes the attractive boy.
 7019 (13.34) The girl doesn't like the boy.
 7020 (13.35) The girl likes her dog.

7021 For (13.32), recall that multi-token names are created using `op1`, `op2`, etc. You will
 7022 need to consult Banerjee et al. (2013) for (13.34), and Schneider et al. (2015) for
 7023 (13.35). You may assume that *her* refers to *the girl* in this example.

- 7024 10. Using an off-the-shelf PropBank SRL system,¹⁷ build a simplified question answer-
 7025 ing system in the style of Shen and Lapata (2007). Specifically, your system should
 7026 do the following:

- 7027 • For each document in a collection, it should apply the semantic role labeler,
 7028 and should store the output as a tuple.
- 7029 • For a question, your system should again apply the semantic role labeler. If
 7030 any of the roles are filled by a *wh*-pronoun, you should mark that role as the
 7031 expected answer phrase (EAP).
- 7032 • To answer the question, search for a stored tuple which matches the question as
 7033 well as possible (same predicate, no incompatible semantic roles, and as many
 7034 matching roles as possible). Align the EAP against its role filler in the stored
 7035 tuple, and return this as the answer.

7036 To evaluate your system, download a set of three news articles on the same topic,
 7037 and write down five factoid questions that should be answerable from the arti-
 7038 cles. See if your system can answer these questions correctly. (If this problem is
 7039 assigned to an entire class, you can build a large-scale test set and compare various
 7040 approaches.)

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/>).

7041 Chapter 14

7042 Distributional and distributed 7043 semantics

7044 A recurring theme in natural language processing is the complexity of the mapping from
7045 words to meaning. In chapter 4, we saw that a single word form, like *bank*, can have mul-
7046 tiple meanings; conversely, a single meaning may be created by multiple surface forms,
7047 a lexical semantic relationship known as **synonymy**. Despite this complex mapping be-
7048 tween words and meaning, natural language processing systems usually rely on words
7049 as the basic unit of analysis. This is especially true in semantics: the logical and frame
7050 semantic methods from the previous two chapters rely on hand-crafted lexicons that map
7051 from words to semantic predicates. But how can we analyze texts that contain words
7052 that we haven't seen before? This chapter describes methods that learn representations
7053 of word meaning by analyzing unlabeled data, vastly improving the generalizability of
7054 natural language processing systems. The theory that makes it possible to acquire mean-
7055 ingful representations from unlabeled data is the **distributional hypothesis**.

7056 14.1 The distributional hypothesis

7057 Here's a word you may not know: *tezgüino*.¹ If you do not know the meaning of *tezgüino*,
7058 then you are in the same situation as a natural language processing system when it en-
7059 counters a word that did not appear in its training data. Now suppose you see that
7060 *tezgüino* is used in the following contexts:

- 7061 • **C1:** *A bottle of _____ is on the table*
- 7062 • **C2:** *Everybody likes _____.*
- 7063 • **C3:** *Don't have _____ before you drive.*

¹The example is from Lin (1998).

	C1	C2	C3	C4	...
<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

- 7064 • C4: *We make _____ out of corn.*

7065 What other words fit into these contexts? How about: *loud*, *motor oil*, *tortillas*, *choices*,
 7066 *wine*? Each row of Table 14.1 is a vector that summarizes the contextual properties for
 7067 each word, with a value of one for contexts in which the word can appear, and a value of
 7068 zero for contexts in which it cannot. Based on these vectors, we can conclude:

- 7069 • *wine* is very similar to *tezgüino*;
 7070 • *motor oil* and *tortillas* are fairly similar to *tezgüino*;
 7071 • *loud* is different.

7072 These vectors, which we will call **word representations**, describe the **distributional**
 7073 properties of each word. Does vector similarity imply semantic similarity? This is the **dis-**
 7074 **distributional hypothesis**, stated by Firth (1957) as: “You shall know a word by the company
 7075 it keeps.” The distributional hypothesis has stood the test of time: distributional statistics
 7076 are a core part of language technology today, because they make it possible to leverage
 7077 large amounts of unlabeled data to learn about rare words that do not appear in labeled
 7078 training data.

7079 Distributional statistics have the striking ability to capture lexical semantic relation-
 7080 ships such as analogies. Figure 14.1 shows two examples, based on two-dimensional
 7081 projections of distributional **word embeddings**, discussed later in this chapter. In each
 7082 case, word-pair relationships correspond to regular linear patterns in this two dimen-
 7083 sional space. No labeled data about the nature of these relationships was required to
 7084 identify this underlying structure.

7085 **Distributional** semantics are computed from context statistics. **Distributed** seman-
 7086 tics are a related but distinct idea: that meaning can be represented by numerical vectors
 7087 rather than symbolic structures. Distributed representations are often estimated from dis-
 7088 tributional statistics, as in latent semantic analysis and WORD2VEC, described later in this

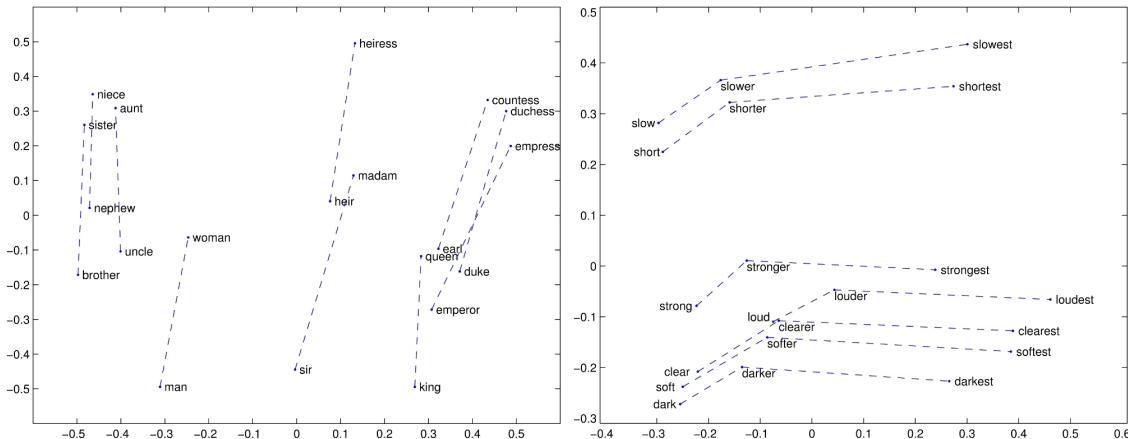


Figure 14.1: Lexical semantic relationships have regular linear structures in two dimensional projections of distributional statistics. From [http://nlp.stanford.edu/projects/glove/.\[todo: redo to make words bigger?\]](http://nlp.stanford.edu/projects/glove/.[todo: redo to make words bigger?])

7089 chapter. However, distributed representations can also be learned in a supervised fashion
 7090 from labeled data, as in the neural classification models encountered in chapter 3.

7091 14.2 Design decisions for word representations

7092 There are many approaches for computing word representations, but most can be dis-
 7093 tinguished on three main dimensions of variation: the nature of the representation, the
 7094 source of contextual information, and the estimation procedure.

7095 14.2.1 Representation

7096 Today, the dominant word representations are k -dimensional vectors of real numbers,
 7097 known as **word embeddings**. (The name is due to the fact that each discrete word is em-
 7098 bedded in a continuous vector space.) This representation dates back at least to the late
 7099 1980s (Deerwester et al., 1990), and is used in popular techniques such as WORD2VEC (Mikolov
 7100 et al., 2013).

7101 Dense vector word embeddings are well suited for neural network architectures, where
 7102 they can be plugged in as inputs. They can also be applied in linear classifiers and struc-
 7103 ture prediction models (Turian et al., 2010), although it can be difficult to learn linear
 7104 models that employ real-valued features (Kummerfeld et al., 2015). A popular alterna-
 7105 tive is bit-string representations, such as **Brown clusters** (§ 14.4), in which each word is
 7106 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, or whether to estimate distinct embeddings for each word sense. 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 by integrating unsupervised clustering with word embedding estimation (Huang and Yates, 2012; Li and Jurafsky, 2015). However, Arora et al. (2016) argue that it is unnecessary to model distinct word senses explicitly, because the embeddings for each surface form are a linear combination of the embeddings of the underlying senses.

14.2.2 Context

The distributional hypothesis says that word meaning is related to the “contexts” in which the word appears, but context can be defined in many ways. In the *tezgiiino* example, contexts are entire sentences, but in practice there are far too many sentences for this to work. At the opposite extreme, the context could be defined as the immediately preceding word; this is the context considered in Brown clusters. WORD2VEC takes an intermediate approach, using local neighborhoods of words (e.g., $h = 5$) as contexts (Mikolov et al., 2013). Contexts can also be much larger: for example, in **latent semantic analysis**, each word’s context vector includes an entry per document, with a value of one if the word appears in the document (Deerwester et al., 1990); in **explicit semantic analysis**, these documents are Wikipedia pages (Gabrilovich and Markovitch, 2007).

Words in context can be labeled by their position with respect to the target word w_m (e.g., two words before, one word after), which makes the resulting word representations more sensitive to syntactic differences (Ling et al., 2015). Another way to incorporate syntax is to perform parsing as a preprocessing step, and then form context vectors from the set of dependency edges (Levy and Goldberg, 2014) or predicate-argument relations (Lin, 1998). The resulting context vectors for several of these methods are shown in Table 14.2.

The choice of context has a profound effect on the resulting representations, which can be viewed in terms of word similarity. Applying latent semantic analysis (§ 14.3) to contexts of size $h = 2$ and $h = 30$ yields the following nearest-neighbors for the word *dog*:²

- ($h = 2$): *cat, horse, fox, pet, rabbit, pig, animal, mongrel, sheep, pigeon*
- ($h = 30$): *kennel, puppy, pet, bitch, terrier, rottweiler, canine, cat, to bark, Alsatian*

Which word list is better? Each word in the $h = 2$ list is an animal, reflecting the fact that locally, the word *dog* tends to appear in the same contexts as other animal types (e.g., *pet*

²The 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/>

The moment one learns English, complications set in.

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)	$\{(one, NSUBJ), (English, DOBJ), (moment, ACL^{-1})\}$

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*.

7140 *the dog, feed the dog*). In the $h = 30$ list, nearly everything is dog-related, including specific
 7141 breeds such as *rottweiler* and *Alsatian*. The list also includes words that are not animals
 7142 (*kennel*), and in one case (*to bark*), is not a noun at all. The 2-word context window is more
 7143 sensitive to syntax, while the 30-word window is more sensitive to topic.

7144 14.2.3 Estimation

7145 Word embeddings are estimated by optimizing some objective: the likelihood of a set of
 7146 unlabeled data (or a closely related quantity), or the reconstruction of a matrix of context
 7147 counts, similar to Table 14.1.

7148 **Maximum likelihood estimation** Likelihood-based optimization is derived from the
 7149 objective $\log p(\mathbf{w}; \mathbf{u})$, where \mathbf{u}_i is the embedding of word i , and $\mathbf{w} = \{\mathbf{w}_m\}_{m=1}^M$ is a cor-
 7150 pus, represented as a list of M tokens. Recurrent neural network language models (§ 6.3)
 7151 optimize this objective directly, backpropagating to the input word embeddings through
 7152 the recurrent structure. However, state-of-the-art word embeddings employ huge cor-
 7153 pora with hundreds of billions of tokens, and recurrent architectures are difficult to scale
 7154 to such data. As a result, likelihood-based word embeddings are usually based on simpli-
 7155 fied 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,

$$\min_{\mathbf{u}, \mathbf{v}} \|\mathbf{C} - \tilde{\mathbf{C}}(\mathbf{u}, \mathbf{v})\|_F, \quad [14.1]$$

where $\tilde{\mathbf{C}}(\mathbf{u}, \mathbf{v})$ is the approximate reconstruction resulting from the embeddings \mathbf{u} and \mathbf{v} , and $\|\mathbf{X}\|_F$ indicates the Frobenius norm, $\sum_{i,j} x_{i,j}^2$. Rather than factoring the matrix of word-context counts, it is useful to transform these counts using information-theoretic metrics such as **pointwise mutual information** (PMI), described in the next section.

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),

$$\min_{\mathbf{U} \in \mathbb{R}^{V \times K}, \mathbf{S} \in \mathbb{R}^{K \times K}, \mathbf{V} \in \mathbb{R}^{|\mathcal{C}| \times K}} \|\mathbf{C} - \mathbf{USV}^\top\|_F \quad [14.2]$$

$$s.t. \quad \mathbf{U}^\top \mathbf{U} = \mathbb{I} \quad [14.3]$$

$$\mathbf{V}^\top \mathbf{V} = \mathbb{I} \quad [14.4]$$

$$\forall i \neq j, \mathbf{S}_{i,j} = 0, \quad [14.5]$$

where V is the size of the vocabulary and $|\mathcal{C}|$ is the number of contexts. The word embeddings can then be set to the rows of the matrix \mathbf{U} . The matrix \mathbf{S} is constrained to be diagonal (these are called the singular values), and the columns of the product \mathbf{SV}^\top provide descriptions of the contexts. Each element $c_{i,j}$ is then reconstructed as a **bilinear product**,

$$c_{i,j} \approx \sum_{k=1}^K u_{i,k} s_k v_{j,k}, \quad [14.6]$$

and the objective is to minimize the sum of squared approximation errors. The orthonormality 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 uncorrelated, so that each dimension conveys unique information. Efficient implementations of truncated singular value decomposition are available in numerical computing packages such as `scipy` and `matlab`.³

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;

³An 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 $V \times |\mathcal{C}|$.

Church and Hanks, 1990), which captures the degree of association between word i and context j ,

$$\text{PMI}(i, j) = \log \frac{p(i, j)}{p(i)p(j)} = \log \frac{p(i | j)p(j)}{p(i)p(j)} = \log \frac{p(i | j)}{p(i)} \quad [14.7]$$

$$= \log \text{count}(i, j) - \log \sum_{i'=1}^V \text{count}(i', j) - \log \sum_{j' \in \mathcal{C}} \text{count}(i, j') + \log \sum_{i'=1}^V \sum_{j' \in \mathcal{C}} \text{count}(i', j'), \quad [14.8]$$

7171 The pointwise mutual information can be viewed as the logarithm of the ratio of the
 7172 conditional probability of word i in context j to the marginal probability of word i in all
 7173 contexts. When word i is statistically associated with context j , the ratio will be greater
 7174 than one, so $\text{PMI}(i, j) > 0$. The PMI transformation focuses latent semantic analysis on re-
 7175 constructing strong word-context associations, rather than on reconstructing large counts.

7176 The PMI is negative when a word and context occur together less often than if they
 7177 were independent, but such negative correlations are unreliable because counts of rare
 7178 events have high variance. Furthermore, the PMI is undefined when $\text{count}(i, j) = 0$. **Pos-**
 7179 **itive PMI (PPMI)** is defined as,

$$\text{PPMI}(i, j) = \begin{cases} \text{PMI}(i, j), & p(i | j) > p(i) \\ 0, & \text{otherwise.} \end{cases} \quad [14.9]$$

7180 Bullinaria and Levy (2007) compare a range of matrix transformations for latent se-
 7181 mantic analysis, using a battery of tasks related to word meaning and word similarity
 7182 (for more on evaluation, see § 14.6). They find that PPMI-based latent semantic analysis
 7183 yields strong performance on a battery of tasks related to word meaning: for example,
 7184 PPMI-based LSA vectors can be used to solve multiple-choice word similarity questions
 7185 from the Test of English as a Foreign Language (TOEFL), obtaining 85% accuracy.

7186 14.4 Brown clusters

7187 Learning algorithms like perceptron and conditional random fields often perform better
 7188 with discrete feature vectors. A simple way to obtain discrete representations from distri-
 7189 butional statistics is by clustering (§ 5.1.1), so that words in the same cluster have similar
 7190 distributional statistics. This can help in downstream tasks, by sharing features between
 7191 all words in the same cluster. However, there is an obvious tradeoff: if the number of clus-
 7192 ters is too small, the words in each cluster will not have much in common; if the number
 7193 of clusters is too large, then the learner will not see enough examples from each cluster to
 7194 generalize.

7195 The solution to this problem is **hierarchical clustering**: using the distributional statis-
 7196 tics to induce a tree-structured representation. Fragments of Brown cluster trees are shown

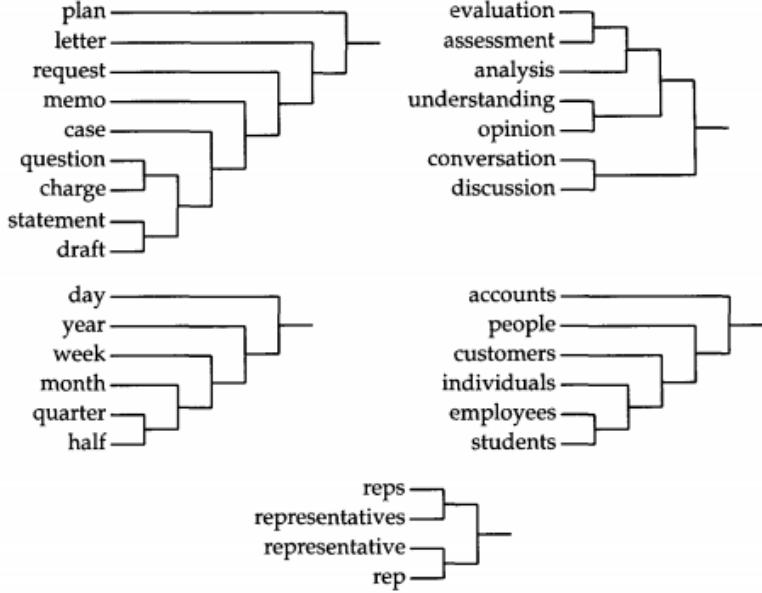


Figure 14.2: Some subtrees produced by bottom-up Brown clustering (Miller et al., 2004) on news text [todo: permission]

7197 in Figure 14.2 and Table 14.3. Each word’s representation consists of a binary string
 7198 describing a path through the tree: 0 for taking the left branch, and 1 for taking the
 7199 right branch. In the subtree in the upper right of the figure, the representation of the
 7200 word *conversation* is 10; the representation of the word *assessment* is 0001. Bitstring pre-
 7201 fixes capture similarity at varying levels of specificity, and it is common to use the first
 7202 eight, twelve, sixteen, and twenty bits as features in tasks such as named entity recogni-
 7203 tion (Miller et al., 2004) 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 :

$$\log p(\mathbf{w}; \mathbf{k}) \approx \sum_{m=1}^M \log p(w_m | w_{m-1}; \mathbf{k}) \quad [14.10]$$

$$\triangleq \sum_{m=1}^M \log p(w_m | k_{w_m}) + \log p(k_{w_m} | k_{w_{m-1}}). \quad [14.11]$$

7204 This is similar to a hidden Markov model, with the crucial difference that each word can
 7205 be emitted from only a single cluster: $\forall k \neq k_{w_m}, p(w_m | k) = 0$.

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.

Using the objective in Equation 14.11, 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 14.11 is maximized. Although the objective seems to involve a sum over the entire corpus, the score for each merger can be computed from the co-occurrence statistics $\text{count}(i, j)$, which counts the number of times a word from cluster i follows a word from cluster j . 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**,

$$I(\mathbf{k}) = \sum_{k_1=1}^K \sum_{k_2=1}^K p(k_1, k_2) \times \text{PMI}(k_1, k_2) \quad [14.12]$$

$$p(k_1, k_2) = \frac{\text{count}(k_1, k_2)}{\sum_{k_1'=1}^K \sum_{k_2'=1}^K \text{count}(k_1', k_2')},$$

7206 where $p(k_1, k_2)$ is the joint probability of a bigram involving a word in cluster k_1 followed
 7207 by a word in k_2 . This probability and the PMI are both computed from the co-occurrence

Algorithm 17 Exchange clustering algorithm

```

procedure EXCHANGECLUSTERING({count(.,.)},  $K$ )
  for  $i \in \mathcal{V}$  do                                 $\triangleright$  Compute unigram counts
     $\text{count}(i) \leftarrow \sum_{j \in \mathcal{C}} \text{count}(i, j)$ 
   $\{w_1, w_2, \dots, w_V\} \leftarrow \text{Sort-Descending}(\{\text{count}(\cdot)\})$ 
  for  $i \in 1 \dots K$  do           $\triangleright$  Put each of the most common words in its own cluster
     $k_i \leftarrow i$ 
  for  $i \in \{K + 1, K + 2, \dots, V\}$  do       $\triangleright$  Iteratively add each word to the clustering
     $k_i \leftarrow K + 1$ 
    Let  $(k, k')$  be the two clusters whose merger maximizes  $I(k)$  (Equation 14.12)
    Renumber clusters so that  $k$  and  $k'$  are merged
    Update cluster co-occurrence counts
  repeat                                 $\triangleright$  Merge the remaining clusters into a tree
    Merge clusters  $(k, k')$  to maximize  $I(k)$ .
  until only one cluster remains
  return Tree structure induced by merge history

```

7208 counts between clusters. After each merger, the co-occurrence vectors for the merged
 7209 clusters are simply added up, so that the next optimal merger can be found efficiently.

7210 This bottom-up procedure requires iterating over the entire vocabulary, and evaluating
 7211 K_t^2 possible mergers at each step, where K_t is the current number of clusters at step t
 7212 of the algorithm. Furthermore, computing the score for each merger involves a sum over
 7213 K_t^2 clusters. The maximum number of clusters is $K_0 = V$, which occurs when every word
 7214 is in its own cluster at the beginning of the algorithm. The time complexity is thus $\mathcal{O}(V^5)$.

7215 To avoid this complexity, practical implementations use a heuristic approximation
 7216 called **exchange clustering**. The K most common words are placed in clusters of their
 7217 own at the beginning of the process. We then consider the next most common word, and
 7218 merge it with one of the existing clusters. This continues until the entire vocabulary has
 7219 been incorporated, at which point the K clusters are merged down to a single cluster,
 7220 forming a tree. The algorithm never considers more than $K + 1$ clusters at any step, and
 7221 the complexity is $\mathcal{O}(V \log V)$ in the size of the vocabulary, bounded by the cost of sorting
 7222 the words at the beginning of the algorithm.

7223 **14.5 Neural word embeddings**

7224 Neural word embeddings combine aspects of the previous two methods: like latent se-
 7225 mantic analysis, they are a continuous vector representation; like Brown clusters, they are
 7226 trained from a likelihood-based objective. Let the vector u_i represent the K -dimensional

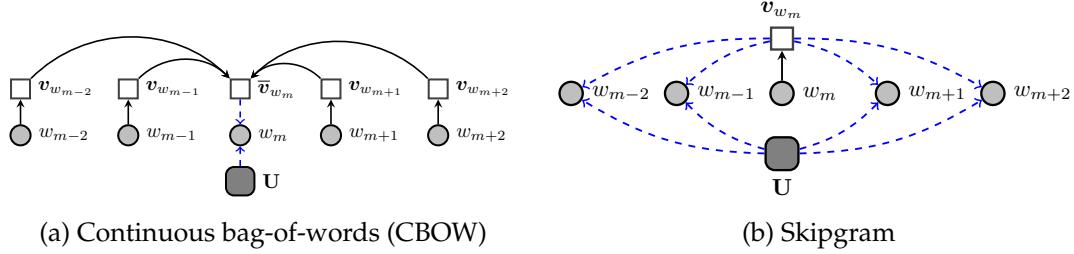


Figure 14.3: The CBOW and skipgram variants of WORD2VEC. The parameter \mathbf{U} is the matrix of word embeddings, and each v_m is the context embedding for word w_m .

embedding for word i , and let v_j represent the K -dimensional embedding for context j . The inner product $u_i \cdot v_j$ represents the compatibility between word i and context j . By incorporating this inner product into an approximation to the log-likelihood of a corpus, it is possible to estimate both parameters by backpropagation. WORD2VEC (Mikolov et al., 2013) includes two such approximations: continuous bag-of-words (CBOW) and skipgrams.

14.5.1 Continuous bag-of-words (CBOW)

In recurrent neural network language models, each word w_m is conditioned on a recurrently updated state vector, which is based on word representations going all the way back to the beginning of the text. The **continuous bag-of-words (CBOW)** model is a simplification: the local context is computed as an average of embeddings for words in the immediate neighborhood $m - h, m - h + 1, \dots, m + h - 1, m + h$,

$$\bar{v}_m = \frac{1}{2h} \sum_{n=1}^h v_{m+n} + v_{m-n}. \quad [14.13]$$

Thus, CBOW is a bag-of-words model, because the order of the context words does not matter; it is continuous, because rather than conditioning on the words themselves, we condition on a continuous vector constructed from the word embeddings. The parameter 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,

$$\log p(\mathbf{w}) \approx \sum_{m=1}^M \log p(w_m | w_{m-h}, w_{m-h+1}, \dots, w_{m+h-1}, w_{m+h}) \quad [14.14]$$

$$= \sum_{m=1}^M \log \frac{\exp(\mathbf{u}_{w_m} \cdot \bar{\mathbf{v}}_m)}{\sum_{j=1}^V \exp(\mathbf{u}_j \cdot \bar{\mathbf{v}}_m)} \quad [14.15]$$

$$= \sum_{m=1}^M \mathbf{u}_{w_m} \cdot \bar{\mathbf{v}}_m - \log \sum_{j=1}^V \exp(\mathbf{u}_j \cdot \bar{\mathbf{v}}_m). \quad [14.16]$$

7243 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:

$$\log p(\mathbf{w}) \approx \sum_{m=1}^M \sum_{n=1}^{h_m} \log p(w_{m-n} | w_m) + \log p(w_{m+n} | w_m) \quad [14.17]$$

$$= \sum_{m=1}^M \sum_{n=1}^{h_m} \log \frac{\exp(\mathbf{u}_{w_{m-n}} \cdot \mathbf{v}_{w_m})}{\sum_{j=1}^V \exp(\mathbf{u}_j \cdot \mathbf{v}_{w_m})} + \log \frac{\exp(\mathbf{u}_{w_{m+n}} \cdot \mathbf{v}_{w_m})}{\sum_{j=1}^V \exp(\mathbf{u}_j \cdot \mathbf{v}_{w_m})} \quad [14.18]$$

$$= \sum_{m=1}^M \sum_{n=1}^{h_m} \mathbf{u}_{w_{m-n}} \cdot \mathbf{v}_{w_m} + \mathbf{u}_{w_{m+n}} \cdot \mathbf{v}_{w_m} - 2 \log \sum_{j=1}^V \exp(\mathbf{u}_j \cdot \mathbf{v}_{w_m}). \quad [14.19]$$

7244 In the skipgram approximation, each word is generated multiple times; each time it is con-
 7245 ditioned only on a single word. This makes it possible to avoid averaging the word vec-
 7246 tors, as in the CBOW model. The local neighborhood size h_m is randomly sampled from
 7247 a uniform categorical distribution over the range $\{1, 2, \dots, h_{\max}\}$; Mikolov et al. (2013) set
 7248 $h_{\max} = 10$. Because the neighborhood grows outward with h , this approach has the effect
 7249 of weighting near neighbors more than distant ones. Skipgram performs better on most
 7250 evaluations than CBOW (see § 14.6 for details of how to evaluate word representations),
 7251 but CBOW is faster to train (Mikolov et al., 2013).

7252 14.5.3 Computational complexity

7253 The WORD2VEC models can be viewed as an efficient alternative to recurrent neural net-
 7254 work language models, which involves a recurrent state update whose time complexity
 7255 is quadratic in the size of the recurrent state vector. CBOW and skipgram avoid this
 7256 computation, and incur only a linear time complexity in the size of the word and con-
 7257 text representations. However, all three models compute a normalized probability over

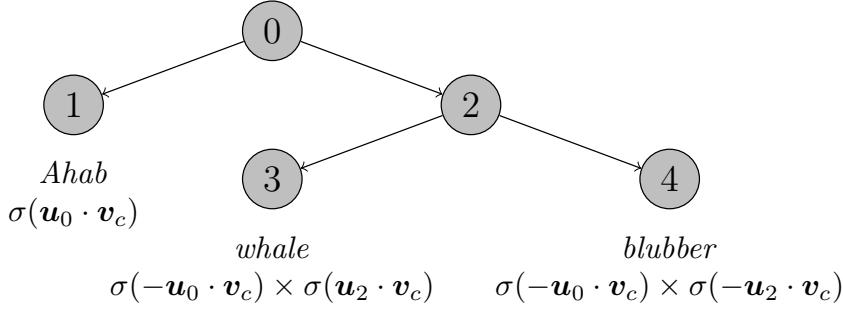


Figure 14.4: A fragment of a hierarchical softmax tree. The probability of each word is computed as a product of probabilities of local branching decisions in the tree.

7258 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
7259 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
7260 sampling**, an approximation that eliminates the dependence on vocabulary size. Both of
7261 these methods are also applicable to RNN language models.
7262

7264 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 ,

$$\Pr(\text{left at } n \mid c) = \sigma(\mathbf{u}_n \cdot \mathbf{v}_c) \quad [14.20]$$

$$\Pr(\text{right at } n \mid c) = 1 - \sigma(\mathbf{u}_n \cdot \mathbf{v}_c) = \sigma(-\mathbf{u}_n \cdot \mathbf{v}_c), \quad [14.21]$$

7265 where σ refers to the sigmoid function, $\sigma(x) = \frac{1}{1+\exp(-x)}$. The range of the sigmoid is the
7266 interval $(0, 1)$, and $1 - \sigma(x) = \sigma(-x)$.

7267 As shown in Figure 14.4, the probability of generating each word is redefined as the
7268 product of the probabilities across its path. The sum of such probabilities is guaranteed to
7269 be one, for any context vector $\mathbf{v}_c \in \mathbb{R}^K$. In a balanced binary tree, the depth is logarithmic
7270 in the number of leaf nodes, and thus the number of multiplications is equal to $\mathcal{O}(\log V)$.
7271 The number of non-leaf nodes is equal to $\mathcal{O}(2V - 1)$, so the number of parameters to be
7272 estimated increases by only a small multiple. The tree can be constructed using an in-
7273 cremental clustering procedure similar to hierarchical Brown clusters (Mnih and Hinton,

7274 2008), or by using the Huffman (1952) encoding algorithm for lossless compression (Huff-
 7275 man, 1952).

7276 **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 difference between the score for the word that was really observed in each context, and the scores for a set of randomly selected **negative samples**:

$$\psi(i, j) = \log \sigma(\mathbf{u}_i \cdot \mathbf{v}_j) + \sum_{i' \in \mathcal{W}_{\text{neg}}} \log(1 - \sigma(\mathbf{u}_{i'} \cdot \mathbf{v}_j)), \quad [14.22]$$

7277 where $\psi(i, j)$ is the score for word i in context j , σ refers to the sigmoid function, and
 7278 \mathcal{W}_{neg} is the set of negative samples. The objective is to maximize the sum over the corpus,
 7279 $\sum_{m=1}^M \psi(w_m, c_m)$, where w_m is token m and c_m is the associated context.

7280 The set of negative samples \mathcal{W}_{neg} is obtained by sampling from a unigram language
 7281 model. Mikolov et al. (2013) construct this unigram language model by exponentiating
 7282 the empirical word probabilities, setting $\hat{p}(i) \propto (\text{count}(i))^{\frac{3}{4}}$. This has the effect of redis-
 7283 tributing probability mass from common to rare words. The number of negative samples
 7284 increases the time complexity of training by a constant factor. Mikolov et al. (2013) report
 7285 that 5-20 negative samples works for small training sets, and that two to five samples
 7286 suffice for larger corpora.

7287 **14.5.4 Word embeddings as matrix factorization**

7288 The negative sampling objective in Equation 14.22 can be justified as an efficient approx-
 7289 imation to the log-likelihood, but it is also closely linked to the matrix factorization ob-
 7290 jective employed in latent semantic analysis. For a matrix of word-context pairs in which
 7291 all counts are non-zero, negative sampling is equivalent to factorization of the matrix M ,
 7292 where $M_{ij} = \text{PMI}(i, j) - \log k$: each cell in the matrix is equal to the pointwise mutual
 7293 information of the word and context, shifted by $\log k$, with k equal to the number of neg-
 7294 ative samples (Levy and Goldberg, 2014). For word-context pairs that are not observed in
 7295 the data, the pointwise mutual information is $-\infty$, but this can be addressed by consid-
 7296 ering only PMI values that are greater than $\log k$, resulting in a matrix of **shifted positive**
 7297 **pointwise mutual information**,

$$M_{ij} = \max(0, \text{PMI}(i, j) - \log k). \quad [14.23]$$

7298 Word embeddings are obtained by factoring this matrix with truncated singular value
 7299 decomposition.

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)).

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, $M_{ij} = \log \text{count}(i, j)$. We then minimize the sum of squares,

$$\begin{aligned} \min_{\mathbf{u}, \mathbf{v}, b, \tilde{b}} \quad & \sum_{j=1}^V \sum_{j \in \mathcal{C}} f(M_{ij}) \left(\widehat{\log M_{ij}} - \log M_{ij} \right)^2 \\ \text{s.t.} \quad & \widehat{\log M_{ij}} = \mathbf{u}_i \cdot \mathbf{v}_j + b_i + \tilde{b}_j, \end{aligned} \quad [14.24]$$

7300 where b_i and \tilde{b}_j are offsets for word i and context j , which are estimated jointly with the
 7301 embeddings \mathbf{u} and \mathbf{v} . The weighting function $f(M_{ij})$ is set to be zero at $M_{ij} = 0$, thus
 7302 avoiding the problem of taking the logarithm of zero counts; it saturates at $M_{ij} = m_{\max}$,
 7303 thus avoiding the problem of overcounting common word-context pairs. This heuristic
 7304 turns out to be critical to the method’s performance.

7305 The time complexity of sparse matrix reconstruction is determined by the number of
 7306 non-zero word-context counts. Pennington et al. (2014) show that this number grows
 7307 sublinearly with the size of the dataset: roughly $\mathcal{O}(N^{0.8})$ for typical English corpora. In
 7308 contrast, the time complexity of WORD2VEC is linear in the corpus size. Computing the co-
 7309 occurrence counts also requires linear time in the size of the corpus, but this operation can
 7310 easily be parallelized using MapReduce-style algorithms (Dean and Ghemawat, 2008).

7311 14.6 Evaluating word embeddings

7312 Distributed word representations can be evaluated in two main ways. **Intrinsic** evalua-
 7313 tions test whether the representations cohere with our intuitions about word meaning.
 7314 **Extrinsic** evaluations test whether they are useful for downstream tasks, such as sequence
 7315 labeling.

7316 **14.6.1 Intrinsic evaluations**

7317 A basic question for word embeddings is whether the similarity of words i and j is re-
 7318 flected in the similarity of the vectors \mathbf{u}_i and \mathbf{u}_j . **Cosine similarity** is typically used to
 7319 compare two word embeddings,

$$\cos(\mathbf{u}_i, \mathbf{u}_j) = \frac{\mathbf{u}_i \cdot \mathbf{u}_j}{\|\mathbf{u}_i\|_2 \times \|\mathbf{u}_j\|_2}. \quad [14.25]$$

7320 For any embedding method, we can then ask whether the cosine similarity of word em-
 7321 beddings is correlated with human judgments of word similarity. The WS-353 dataset (Finkel-
 7322 stein et al., 2002) includes similarity scores for 353 word pairs (Table 14.4). To test the
 7323 accuracy of embeddings for rare and morphologically complex words, Luong et al. (2013)
 7324 introduce a dataset of “rare words.” Outside of English, word similarity resources are
 7325 limited, mainly consisting of translations of WS-353.

7326 Word analogies (e.g., *king:queen :: man:woman*) have also been used to evaluate word
 7327 embeddings (Mikolov et al., 2013). In this evaluation, the system is provided with the first
 7328 three parts of the analogy ($i_1 : j_1 :: i_2 : ?$), and the final element is predicted by finding the
 7329 word embedding most similar to $\mathbf{u}_{i_1} - \mathbf{u}_{j_1} + \mathbf{u}_{i_2}$. Another evaluation tests whether word
 7330 embeddings are related to broad lexical semantic categories called **supersenses** (Ciaramita
 7331 and Johnson, 2003): verbs of motion, nouns that describe animals, nouns that describe
 7332 body parts, and so on. These supersenses are annotated for English synsets in Word-
 7333 Net (Fellbaum, 2010). This evaluation is implemented in the `qvec` metric, which tests
 7334 whether the matrix of supersenses can be reconstructed from the matrix of word embed-
 7335 dings (Tsvetkov et al., 2015).

7336 Levy et al. (2015) compared several dense word representations for English — includ-
 7337 ing latent semantic analysis, WORD2VEC, and GloVe — using six word similarity metrics
 7338 and two analogy tasks. None of the embeddings outperformed the others on every task,
 7339 but skipgrams were the most broadly competitive. Hyperparameter tuning played a key
 7340 role: any method will perform badly if the wrong hyperparameters are used. Relevant
 7341 hyperparameters include the embedding size, as well as algorithm-specific details such
 7342 as the neighborhood size and the number of negative samples.

7343 **14.6.2 Extrinsic evaluations**

7344 Word representations contribute to downstream tasks like sequence labeling and docu-
 7345 ment classification by enabling generalization across words. The use of distributed repres-
 7346 entations as features can be seen as a form of **semi-supervised learning**, in which perfor-
 7347 mance on a supervised learning problem is augmented by learning distributed repres-
 7348 entations from unlabeled data (Miller et al., 2004; Koo et al., 2008; Turian et al., 2010). These
 7349 **pre-trained word representations** can be used as features in a linear prediction model,
 7350 or as the input layer in a neural network, such as a Bi-LSTM tagging model (§ 7.6). Word

representations can be evaluated by the performance of the downstream systems that consume them: for example, GloVe embeddings are convincingly better than Latent Semantic Analysis as features in the downstream task of named entity recognition (Pennington et al., 2014). Unfortunately, extrinsic and intrinsic evaluations do not always point in the same direction, and the best word representations for one downstream task may perform poorly on another task (Schnabel et al., 2015).

When word representations are updated from labeled data in the downstream task, they are said to be **fine-tuned**. When labeled data is plentiful, pre-training may be unnecessary; when labeled data is scarce, fine-tuning may lead to overfitting. Various combinations of pre-training and fine-tuning can be employed. Pre-trained embeddings can be used as initialization before fine-tuning, and this can substantially improve performance (Lample et al., 2016). Alternatively, both fine-tuned and pre-trained embeddings can be used as inputs in a single model (Kim, 2014).

14.7 Distributed representations beyond distributional statistics

Distributional word representations can be estimated from huge unlabeled datasets, thereby covering many words that do not appear in labeled data: for example, GloVe embeddings are estimated from 800 billion tokens of web data,⁴ while the largest labeled datasets for NLP tasks are on the order of millions of tokens. Nonetheless, even a dataset of hundreds of billions of tokens will not cover every word that may be encountered in the future. Furthermore, many words will appear only a small number of times, making their embeddings unreliable. Many languages have more morphological structure than English, and thus have lower token-to-type ratios. When this problem is coupled with small training corpora, it becomes especially important to leverage other sources of information beyond distributional statistics.

14.7.1 Word-internal structure

One solution is to incorporate word-internal structure into word embeddings. Purely distributional approaches consider words as atomic units, but in fact, many words have internal structure, so that their meaning can be **composed** from the representations of sub-word units. Consider the following terms, all of which are missing from Google’s pre-trained WORD2VEC embeddings:⁵

millicuries This word has **morphological** structure (see § 9.1.2 for more on morphology): the prefix *milli-* indicates an amount, and the suffix *-s* indicates a plural. (A *millicurie* is an unit of radioactivity.)

⁴<http://commoncrawl.org/>

⁵<https://code.google.com/archive/p/word2vec/>, accessed September 20, 2017

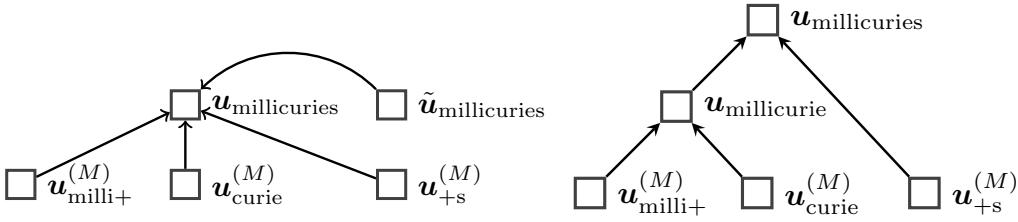


Figure 14.5: Two architectures for building word embeddings from subword units. On the left, morpheme embeddings $u^{(m)}$ are combined by addition with the non-compositional word embedding \tilde{u} (Botha and Blunsom, 2014). On the right, morpheme embeddings are combined in a recursive neural network (Luong et al., 2013).

7384 **caesium** This word is a single morpheme, but the characters *-ium* are often associated
 7385 with chemical elements. (*Caesium* is the British spelling of a chemical element,
 7386 spelled *cesium* in American English.)

7387 **IAEA** This term is an acronym, as suggested by the use of capitalization. The prefix *I-* frequently
 7388 refers to international organizations, and the suffix *-A* often refers to agencies or associations. (*IAEA* is the International Atomic Energy Agency.)

7390 **Zhezhgan** This term is in title case, suggesting the name of a person or place, and the
 7391 character bigram *zh* indicates that it is likely a transliteration. (*Zhezhgan* is a mining
 7392 facility in Kazakhstan.)

7393 How can word-internal structure be incorporated into word representations? One
 7394 approach is to construct word representations from embeddings of the characters or mor-
 7395 phemes. For example, if word i has morphological segments M_i , then its embedding can
 7396 be constructed by addition (Botha and Blunsom, 2014),

$$\mathbf{u}_i = \tilde{\mathbf{u}}_i + \sum_{j \in M_i} \mathbf{u}_j^{(M)}, \quad [14.26]$$

7397 where $\mathbf{u}_m^{(M)}$ is a morpheme embedding and $\tilde{\mathbf{u}}_i$ is a non-compositional embedding of the
 7398 whole word, which is an additional free parameter of the model (Figure 14.5, left side).
 7399 All embeddings are estimated from a **log-bilinear language model** (Mnih and Hinton,
 7400 2007), which is similar to the CBOW model (§ 14.5), but includes only contextual informa-
 7401 tion from preceding words. The morphological segments are obtained using an unsuper-
 7402 vised segmenter (Creutz and Lagus, 2007). For words that do not appear in the training
 7403 data, the embedding can be constructed directly from the morphemes, assuming that each
 7404 morpheme appears in some other word in the training data. The free parameter $\tilde{\mathbf{u}}$ adds
 7405 flexibility: words with similar morphemes are encouraged to have similar embeddings,
 7406 but this parameter makes it possible for them to be different.

7407 Subword information can be incorporated in various ways:

7408 **Subword units** Examples like *IAEA* and *Zhezhgan* are not based on morphological composition, and a morphological segmenter is unlikely to identify meaningful subword units for these terms. Rather than using morphemes for subword embeddings, one can use characters (Santos and Zadrozny, 2014; Ling et al., 2015; Kim et al., 2016), character n -grams (Wieting et al., 2016; Bojanowski et al., 2017), and **byte-pair encodings**, a compression technique which captures frequent substrings (Gage, 1994; Sennrich et al., 2016).

7415 **Composition** Combining the subword embeddings by addition does not differentiate between orderings, nor does it identify any particular morpheme as the **root**. A range of more flexible compositional models have been considered, including recurrence (Ling et al., 2015), convolution (Santos and Zadrozny, 2014; Kim et al., 2016), and **recursive neural networks** (Luong et al., 2013), in which representations of progressively larger units are constructed over a morphological parse, e.g. $((milli+curie)+s)$, $((in+flam)+able)$, $(in+(vis+ible))$. A recursive embedding models is shown in the right panel of Figure 14.5.

7423 **Estimation** Estimating subword embeddings from a full dataset is computationally expensive. An alternative approach is to train a subword model to match pre-trained word embeddings (Cotterell et al., 2016; Pinter et al., 2017). To train such a model, it is only necessary to iterate over the vocabulary, and the not the corpus.

7427 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, such as synonymy (Faruqui et al., 2015). This is done by minimizing the following objective,

$$\min_{\mathbf{U}} \sum_{j=1}^V \|\mathbf{u}_i - \hat{\mathbf{u}}_i\|_2 + \sum_{(i,j) \in \mathcal{L}} \beta_{ij} \|\mathbf{u}_i - \mathbf{u}_j\|_2, \quad [14.27]$$

7428 where $\hat{\mathbf{u}}_i$ is the pretrained embedding of word i , and $\mathcal{L} = \{(i, j)\}$ is a lexicon of word relations. The hyperparameter β_{ij} controls the importance of adjacent words having similar embeddings; Faruqui et al. (2015) set it to the inverse of the degree of word i , $\beta_{ij} = |\{j : (i, j) \in \mathcal{L}\}|^{-1}$. Retrofitting improves performance on a range of intrinsic evaluations, and gives small improvements on an extrinsic document classification task.

7433 14.8 Distributed representations of multiword units

7434 Can distributed semantics extended to phrases, sentences, paragraphs, and beyond? Be-
 7435 fore exploring this possibility, recall the distinction between distributed and distributional
 7436 representations. Neural embeddings such as WORD2VEC are both distributed (vector-
 7437 based) and distributional (derived from counts of words in context). As we consider larger
 7438 units of text, the counts in any corpus must decrease: in the limit, a multi-paragraph span
 7439 of text would never appear twice, except in cases of plagiarism. Thus, the meaning of
 7440 a large span of text cannot be determined from distributional statistics alone; it must be
 7441 computed compositionally from smaller spans. But these considerations are orthogonal
 7442 to the question of whether distributed representations — dense numerical vectors — are
 7443 sufficiently expressive to capture the meaning of phrases, sentences, and paragraphs.

7444 14.8.1 Purely distributional methods

7445 Some multiword phrases are non-compositional: the meaning of such phrases is not de-
 7446 rived from the meaning of the individual words using typical compositional semantics.
 7447 This includes proper nouns like *San Francisco* as well as idiomatic expressions like *kick*
 7448 *the bucket* (Baldwin and Kim, 2010). For these cases, purely distributional approaches
 7449 can work. A simple approach is to identify multiword units that appear together fre-
 7450 quently, and then treat these units as words, learning embeddings using a technique such
 7451 as WORD2VEC. The problem of identifying multiword units is sometimes called **colloca-**
 7452 **tion extraction**, and can be approached using metrics such as pointwise mutual informa-
 7453 tion: two-word units are extracted first, and then larger units are extracted. Mikolov et al.
 7454 (2013) identify such units and then treat them as words when estimating skipgram em-
 7455 beddings, showing that the resulting embeddings perform reasonably on a task of solving
 7456 phrasal analogies, e.g. *New York : New York Times :: Baltimore : Baltimore Sun*.

7457 14.8.2 Distributional-compositional hybrids

7458 To move beyond short multiword phrases, composition is necessary. A simple but surpris-
 7459 ingly powerful approach is to represent a sentence with the average of its word embed-
 7460 dings (Mitchell and Lapata, 2010). This can be considered a hybrid of the distributional
 7461 and compositional approaches to semantics: the word embeddings are computed distri-
 7462 butionally, and then the sentence representation is computed by composition. Baroni and
 7463 Zamparelli (2010) embed short phrases by treating adjectives as matrices, which operate
 7464 by multiplication on nouns.

7465 The WORD2VEC approach can be stretched considerably further, embedding entire
 7466 sentences using a model similar to skipgrams, in the “skip-thought” model of Kiros et al.
 7467 (2015). Each sentence is **encoded** into a vector using a recurrent neural network: the
 7468 encoding of sentence t is set to the RNN hidden state at its final token, $h_{M_t}^{(t)}$. This vector is

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)

7469 then a parameter in a **decoder** model that is used to generate the previous and subsequent
 7470 sentences: the decoder is another recurrent neural network, which takes the encoding
 7471 of the neighboring sentence as an additional parameter in its recurrent update. (This
 7472 **encoder-decoder model** is discussed at length in chapter 18.) The encoder and decoder
 7473 are trained simultaneously from a likelihood-based objective, and the trained encoder
 7474 can then be used to compute a distributed representation of any sentence. Skip-thought
 7475 can be viewed as a hybrid of distributional and compositional approaches: the vector
 7476 representation of each sentence is computed compositionally from the representations of
 7477 the individual words, but the training objective is distributional, based on sentence co-
 7478 occurrence across a corpus.

7479 **Autoencoders** are a variant of encoder-decoder models in which the decoder is trained
 7480 to produce the same text that was originally encoded, using only the distributed encod-
 7481 ing vector (Li et al., 2015). By interpolating between distributed representations of two
 7482 sentences, $\alpha \mathbf{u}_i + (1 - \alpha) \mathbf{u}_j$, it is possible to generate sentences that combine aspects of
 7483 the two inputs, as shown in Figure 14.6 (Bowman et al., 2016). Autoencoders can also
 7484 be applied to longer texts, such as paragraphs and documents. This enables applications
 7485 such as **question answering**, which can be performed by matching the encoding of the
 7486 question with encodings of candidate answers (Miao et al., 2016).

7487 14.8.3 Supervised compositional methods

7488 Given a supervision signal, such as a label describing the sentiment or meaning of a sen-
 7489 tence, a wide range of compositional methods can be applied to compute a distributed
 7490 representation that can then be used to predict the label. The simplest is to average the
 7491 embeddings of each word in the sentence, and then pass this average through a feed-

7492 forward neural network (Iyyer et al., 2015). Convolutional and recurrent neural networks
 7493 can effectively capture multiword phenomena such as negation (Kalchbrenner et al., 2014;
 7494 Kim, 2014; Li et al., 2015; Tang et al., 2015). Another approach is to incorporate the syn-
 7495 tactic structure of the sentence. **Recursive neural networks** construct a representation for
 7496 each syntactic constituent by operating on the representations of the children, thus prop-
 7497 agating the distributed representation up the tree (Socher et al., 2012). However, Li et al.
 7498 (2015) show that in many cases, recurrent neural networks perform as well or better than
 7499 recursive networks.

7500 Whether convolutional, recurrent, or recursive, a key question is whether supervised
 7501 sentence representations are task-specific, or whether a single supervised sentence repre-
 7502 sentation model can yield useful performance on other tasks. Wieting et al. (2015) train a
 7503 variety of sentence embedding models for the task of labeling pairs of sentences as **para-**
 7504 **phrases**. They show that the resulting sentence embeddings give good performance for
 7505 sentiment analysis. The **Stanford Natural Language Inference corpus** classifies sentence
 7506 pairs as **entailments** (the truth of sentence i implies the truth of sentence j), **contradictions**
 7507 (the truth of sentence i implies the falsity of sentence j), and neutral (i neither entails nor
 7508 contradicts j). Sentence embeddings trained on this dataset transfer to a wide range of
 7509 classification tasks (Conneau et al., 2017).

7510 14.8.4 Hybrid distributed-symbolic representations

7511 The power of distributed representations is in their generality: the distributed represen-
 7512 tation of a unit of text can serve as a summary of its meaning, and therefore as the input
 7513 for downstream tasks such as classification, matching, and retrieval. For example, dis-
 7514 tributed sentence representations can be used to recognize the paraphrase relationship
 7515 between closely related sentences like the following:

- 7516 (14.1) Donald thanked Vlad profusely.
- 7517 (14.2) Donald conveyed to Vlad his profound appreciation.
- 7518 (14.3) Vlad was showered with gratitude by Donald.

7519 Symbolic representations are relatively brittle to this sort of variation, but are better
 7520 suited to describe individual entities, the things that they do, and the things that are done
 7521 to them. In examples (14.1)-(14.3), we not only know that somebody thanked someone
 7522 else, but we can make a range of inferences about what has happened between the en-
 7523 tities named *Donald* and *Vlad*. Because distributed representations do not treat entities
 7524 symbolically, they lack the ability to reason about the roles played by entities across a sen-
 7525 tence or larger discourse.⁶ A hybrid between distributed and symbolic representations

⁶At 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!”

7526 might give the best of both worlds: robustness to the many different ways of describing
7527 the same event, plus the expressiveness to support inferences about entities and the roles
7528 that they play.

7529 A “top-down” hybrid approach is to begin with logical semantics (of the sort described
7530 in the previous two chapters), and then relax the requirement that a predefined lexicon
7531 associate words with each of a large set of predefined semantic predicates: instead, pred-
7532 icates are identified with distributional word clusters (Poon and Domingos, 2009; Lewis
7533 and Steedman, 2013). A “bottom-up” approach is to add minimal symbolic structure to
7534 existing distributed representations, such as vector representations for each entity (Ji and
7535 Eisenstein, 2015; Wiseman et al., 2016). This has been shown to improve performance on
7536 two problems that we will encounter in the following chapters: classification of **discourse**
7537 **relations** between adjacent sentences (chapter 16; Ji and Eisenstein, 2015), and **coreference**
7538 **resolution** of entity mentions (chapter 15; Wiseman et al., 2016; Ji et al., 2017). Research on
7539 hybrid semantic representations is still in an early stage, and future representations may
7540 deviate more boldly from existing symbolic and distributional approaches.

7541 Additional reading

7542 Turney and Pantel (2010) survey a number of facets of vector word representations, fo-
7543 cusing on matrix factorization methods. Schnabel et al. (2015) highlight problems with
7544 similarity-based evaluations of word embeddings, and present a novel evaluation that
7545 controls for word frequency. Baroni et al. (2014) address linguistic issues that arise in
7546 attempts to combine distributed and compositional representations.

7547 In bilingual and multilingual distributed representations, embeddings are estimated
7548 for translation pairs or tuples, such as (*dog, perro, chien*). These embeddings can improve
7549 machine translation (Zou et al., 2013; Klementiev et al., 2012), transfer natural language
7550 processing models across languages (Täckström et al., 2012), and make monolingual word
7551 embeddings more accurate (Faruqui and Dyer, 2014). A typical approach is to learn a pro-
7552 jection that maximizes the correlation of the distributed representations of each element
7553 in a translation pair, which can be obtained from a bilingual dictionary. Distributed rep-
7554 resentations can also be linked to perceptual information, such as image features. Bruni
7555 et al. (2014) use textual descriptions of images to obtain visual contextual information for
7556 various words, which supplements traditional distributional context. Image features can
7557 also be inserted as contextual information in log bilinear language models (Kiros et al.,
7558 2014), making it possible to automatically generate text descriptions of images.

7559 **Exercises**

- 7560 1. Prove that the sum of probabilities of paths through a hierarchical softmax tree is
 7561 equal to one.
2. In skipgram word embeddings, the negative sampling objective can be written as,

$$\mathcal{L} = \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{C}} \text{count}(i, j) \psi(i, j), \quad [14.28]$$

7562 with $\psi(i, j)$ is defined in Equation 14.22.

7563 Suppose we draw the negative samples from the empirical unigram distribution
 7564 $\hat{p}(i) = p_{\text{unigram}}(i)$. First, compute the expectation of \mathcal{L} with respect this probability.

7565 Next, take the derivative of this expectation with respect to the score of a single word
 7566 context pair $\sigma(\mathbf{u}_i \cdot \mathbf{v}_j)$, and solve for the pointwise mutual information $\text{PMI}(i, j)$. You
 7567 should be able to show that at the optimum, the PMI is a simple function of $\sigma(\mathbf{u}_i \cdot \mathbf{v}_j)$
 7568 and the number of negative samples.

- 7569 3. * In Brown clustering, prove that the cluster merge that maximizes the average mu-
 7570 tual information (Equation 14.12) also maximizes the log-likelihood objective (Equa-
 7571 tion 14.11).

7572 For the next two problems, download a set of pre-trained word embeddings, such as the
 7573 WORD2VEC or polyglot embeddings.

- 7574 4. Use cosine similarity to find the most similar words to: *dog, whale, before, however,*
 7575 *fabricate*.
- 7576 5. Use vector addition and subtraction to compute target vectors for the analogies be-
 7577 low. After computing each target vector, find the top three candidates by cosine
 7578 similarity.
- 7579 • *dog:puppy :: cat: ?*
 - 7580 • *speak:speaker :: sing: ?*
 - 7581 • *France:French :: England: ?*
 - 7582 • *France:wine :: England: ?*

7583 The remaining problems will require you to build a classifier and test its properties. Pick
 7584 a multi-class text classification dataset, such as RCV1⁷). Divide your data into training
 7585 (60%), development (20%), and test sets (20%), if no such division already exists.

⁷http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm

- 7586 6. Train a convolutional neural network, with inputs set to pre-trained word embed-
7587 dings from the previous problem. Use a special, fine-tuned embedding for out-of-
7588 vocabulary words. Train until performance on the development set does not im-
7589 prove. You can also use the development set to tune the model architecture, such
7590 as the convolution width and depth. Report *F-MEASURE* and accuracy, as well as
7591 training time.
- 7592 7. Now modify your model from the previous problem to fine-tune the word embed-
7593 dings. Report *F-MEASURE*, accuracy, and training time.
- 7594 8. Try a simpler approach, in which word embeddings in the document are averaged,
7595 and then this average is passed through a feed-forward neural network. Again, use
7596 the development data to tune the model architecture. How close is the accuracy to
7597 the convolutional networks from the previous problems?

7598

Chapter 15

7599

Reference Resolution

7600 References are one of the most noticeable forms of linguistic ambiguity, afflicting not just
7601 automated natural language processing systems, but also fluent human readers. Warnings
7602 to avoid “ambiguous pronouns” are ubiquitous in manuals and tutorials on writing
7603 style. But referential ambiguity is not limited to pronouns, as shown in the text in Fig-
7604 ure 15.1. Each of the underlined substrings in the passage refers to an entity that is in-
7605 troduced earlier in the story. These references include the pronouns *he* and *his*, but also
7606 the shortened name *Cook*, and most challengingly, **nominals** such as *the firm* and *the firm’s*
7607 *biggest growth market*.

7608 **Reference resolution** subsumes several subtasks. This chapter will focus on **corefer-
7609 ence resolution**, which is the task of grouping spans of text that refer to a single underly-
7610 ing entity, or, in some cases, a single event: for example, the spans *Tim Cook*, *he*, and *Cook*
7611 are all **coreferent**. These individual spans are called **mentions**, because they mention an
7612 entity; the entity is sometimes called the **referent**. Each mention has a set of **antecedents**,
7613 which are preceding mentions that are coreferent; for the first mention of an entity, the
7614 antecedent set is empty. The task of **pronominal anaphora resolution** requires only iden-
7615 tifying the antecedents of pronouns. In **entity linking**, references are resolved not to other
7616 spans of text, but to entities in a knowledge base. This task is discussed in chapter 17.

7617 Coreference resolution is a challenging problem for several reasons. Resolving differ-
7618 ent types of **referring expressions** requires different types of reasoning: the features and
7619 methods that are useful for resolving pronouns are different from those that are useful
7620 to resolve names and **nominals** (e.g., *the firm*, *government officials*). Coreference resolution
7621 involves not only linguistic reasoning, but also world knowledge and pragmatics: you
7622 may not have known that *China* was *Apple’s biggest growth market*, but it is likely that you
7623 effortlessly resolved this reference while reading the passage in Figure 15.1.¹ A further

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).

- (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.

7624 challenge is that coreference resolution decisions are often entangled: each mention adds
 7625 information about the entity, which affects other coreference decisions. This means that
 7626 coreference resolution must be addressed as a structure prediction problem. But as we
 7627 will see, there is no dynamic program that allows the space of coreference decisions to be
 7628 searched efficiently.

7629 15.1 Forms of referring expressions

7630 There are three main forms of referring expressions — pronouns, names, and nominals.

7631 15.1.1 Pronouns

7632 Pronouns are a closed class of words that are used for references. A natural way to think
 7633 about pronoun resolution is SMASH (Kehler, 2007):

- 7634 • Search for candidate antecedents;
- 7635 • Match against hard agreement constraints;
- 7636 • And Select using Heuristics, which are “soft” constraints such as recency, syntactic
 7637 prominence, and parallelism.

7638 15.1.1.1 Search

7639 In the search step, candidate antecedents are identified from the preceding text or speech.²
 7640 Any noun phrase can be a candidate antecedent, and pronoun resolution usually requires

Pragmatics is the discipline of linguistics concerned with the formalization of such assumptions (Huang, 2015).

²Pronouns whose referents come later are known as **cataphora**, as in this example from Márquez (1970):

- (15.1) Many years later, as [he] faced the firing squad, [Colonel Aureliano Buendía] was to remember that distant afternoon when his father took him to discover ice.

7641 parsing the text to identify all such noun phrases.³ Filtering heuristics can help to prune
 7642 the search space to noun phrases that are likely to be coreferent (Lee et al., 2013; Durrett
 7643 and Klein, 2013). In nested noun phrases, mentions are generally considered to be the
 7644 largest unit with a given head word: thus, *Apple Inc. Chief Executive Tim Cook* would be
 7645 included as a mention, but *Tim Cook* would not, since they share the same head word,
 7646 *Cook*.

7647 15.1.1.2 Matching constraints for pronouns

7648 References and their antecedents must agree on morphological features such as number,
 7649 person, gender, and animacy. Consider the pronoun *he* in this passage from the running
 7650 example:

- 7651 (15.2) Tim Cook has jetted in for talks with officials as [he] seeks to clear up a pile of
 7652 problems...

7653 The pronoun and possible antecedents have the following features:

- 7654 • *he*: singular, masculine, animate, third person
- 7655 • *officials*: plural, animate, third person
- 7656 • *talks*: plural, inanimate, third person
- 7657 • *Tim Cook*: singular, masculine, animate, third person

7658 The SMASH method searches backwards from *he*, discarding *officials* and *talks* because they
 7659 do not satisfy the agreements constraints.

7660 Another source of constraints comes from syntax — specifically, from the phrase struc-
 7661 ture trees discussed in chapter 10. Consider a parse tree in which both *x* and *y* are phrasal
 7662 constituents. The constituent *x* **c-commands** the constituent *y* iff the first branching node
 7663 above *x* also dominates *y*. For example, in Figure 15.2a, *Abigail* c-commands *her*, because
 7664 the first branching node above *Abigail*, *S*, also dominates *her*. Now, if *x* c-commands *y*,
 7665 **government and binding theory** (Chomsky, 1982) states that *y* can only refer to *x* if it is
 7666 a **reflexive pronoun** (e.g., *herself*). Furthermore, if *y* is a reflexive pronoun, then its an-
 7667 tecedent must c-command it. Thus, in Figure 15.2a, *her* cannot refer to *Abigail*; conversely,
 7668 if we replace *her* with *herself*, then the reflexive pronoun must refer to *Abigail*, since this is
 7669 the only candidate antecedent that c-commands it.

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.

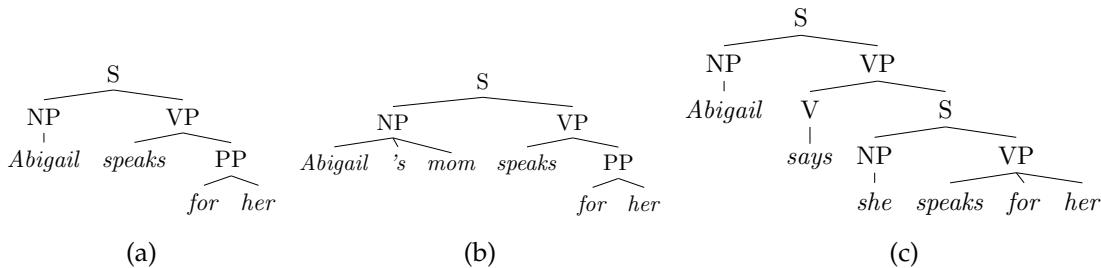


Figure 15.2: In (a), *Abigail* c-commands *her*; in (b), *Abigail* does not c-command *her*, but *Abigail's mom* does; in (c), the scope of *Abigail* is limited by the S non-terminal, so that *she* or *her* can bind to *Abigail*, but not both.

Now consider the example shown in Figure 15.2b. Here, *Abigail* does not c-command *her*, but *Abigail's mom* does. Thus, *her* can refer to *Abigail* — and we cannot use reflexive *herself* in this context, unless we are talking about *Abigail's mom*. However, *her* does not have to refer to *Abigail*. Finally, Figure 15.2c shows how these constraints are limited. In this case, the pronoun *she* can refer to *Abigail*, because the S non-terminal puts *Abigail* outside the domain of *she*. Similarly, *her* can also refer to *Abigail*. But *she* and *her* cannot be coreferent, because *she* c-commands *her*.

7677 15.1.1.3 Heuristics

After applying constraints, heuristics are applied to select among the remaining candidates. Recency is a particularly strong heuristic. All things equal, readers will prefer the more recent referent for a given pronoun, particularly when comparing referents that occur in different sentences. Jurafsky and Martin (2009) offer the following example:

- 7682 (15.3) The doctor found an old map in the captain's chest. Jim found an even older map
7683 hidden on the shelf. [It] described an island.

⁷⁶⁸⁴ Readers are expected to prefer the second, older map as the referent for the pronoun *it*.

However, subjects are often preferred over objects, and this can contradict the preference for recency when two candidate referents are in the same sentence. For example,

- 7687 (15.4) Asha loaned Mei a book on Spanish. [She] is always trying to help people.

7688 Here, we may prefer to link *she* to *Asha* rather than *Mei*, because of *Asha*'s position in the
7689 subject role of the preceding sentence. (Arguably, this preference would not be strong
7690 enough to select *Asha* if the second sentence were *She is visiting Argentina next month.*)

7691 A third heuristic is parallelism:

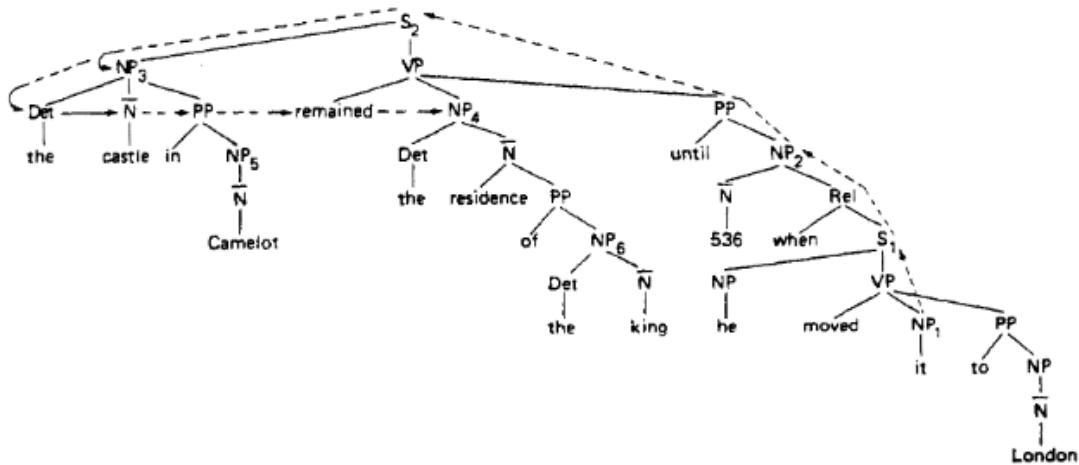


Fig. 2.

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*.[todo: re-make figure in TiKZ.]

7692 (15.5) Asha loaned Mei a book on Spanish. Olya loaned [her] a book on Portuguese.

7693 Here *Mei* is preferred as the referent for *her*, contradicting the preference for the subject
7694 *Asha* in the preceding sentence.

7695 The recency and subject role heuristics can be unified by traversing the document in
7696 a syntax-driven fashion (Hobbs, 1978): each preceding sentence is traversed breadth-first,
7697 left-to-right (Figure 15.3). This heuristic successfully handles (15.4): *Asha* is preferred as
7698 the referent for *she* because the subject NP is visited first. It also handles (15.3): the older
7699 map is preferred as the referent for *it* because the more recent sentence is visited first. (An
7700 alternative unification of recency and syntax is proposed by **centering theory** (Grosz et al.,
7701 1995), which is discussed in detail in chapter 16.)

7702 In early work on reference resolution, the number of heuristics was small enough that
7703 a set of numerical weights could be set by hand (Lappin and Leass, 1994). More recent
7704 work uses machine learning to quantify the importance of each of these factors. How-
7705 ever, pronoun resolution cannot be completely solved by morphological constraints and
7706 syntactic heuristics alone. This is shown by the classic example pair (Winograd, 1972):

7707 (15.6) *The [city council] denied the protesters a permit because [they] feared violence.*

7708 (15.7) *The city council denied [the protesters] a permit because [they] advocated violence.*

7709 **15.1.1.4 Non-referential pronouns**

7710 While pronouns are generally used for reference, they need not refer to entities. The fol-
7711 lowing examples show how pronouns can refer to propositions, events, and speech acts.

7712 (15.8) They told me that I was too ugly for show business, but I didn't believe [it].

7713 (15.9) Asha saw Babak get angry, and I saw [it] too.

7714 (15.10) Asha said she worked in security. I suppose [that]'s one way to put it.

7715 These forms of reference are generally not annotated in coreference resolution datasets
7716 such as OntoNotes (Pradhan et al., 2011).

7717 Pronouns may also have **generic referents**:

7718 (15.11) A poor carpenter blames [her] tools.

7719 (15.12) On the moon, [you] have to carry [your] own oxygen.

7720 (15.13) Every farmer who owns a donkey beats [it]. (Geach, 1962)

7721 In the OntoNotes dataset, coreference is not annotated for generic referents, even in cases
7722 like these examples, in which the same generic entity is mentioned multiple times.

7723 Some pronouns do not refer to anything at all:

7724 (15.14) *[It]'s raining.*

[Il] pleut. (Fr)

7725 (15.15) [It] 's money that she's really after.

7726 (15.16) [It] is too bad that we have to work so hard.

7727 In the first example, *it* and *il* are **pleonastic**. The second and third examples are **cleft** and
7728 **extraposition**.[todo: explain]

7729 How can we automatically distinguish these usages of *it* from referential pronouns?

7730 Consider the the difference between the following two examples (Bergsma et al., 2008):

7731 (15.17) You can make [it] in advance.

7732 (15.18) You can make [it] in showbiz.

7733 In the second example, the pronoun *it* is non-referential. One way to see this is by substi-
7734 tuting another pronoun, like *them*, into these examples:

7735 (15.19) You can make [them] in advance.

7736 (15.20) ? You can make [them] in showbiz.

7737 The questionable grammaticality of the second example suggests that *it* is not referential.
7738 Bergsma et al. (2008) operationalize this idea by comparing distributional statistics for the
7739 *n*-grams around the word *it*, testing how often other pronouns or nouns ever appear in
7740 the same position as *it*. In cases where other pronouns are infrequent, the *it* is unlikely to
7741 be referential.

7742 15.1.2 Proper Nouns

7743 If a proper noun is used as a referring expression, it often refers to another proper noun,
7744 so that the coreference problem is simply to determine whether the two names match.
7745 Subsequent proper noun references often use a shortened form, as in the running example
7746 (Figure 15.1):

7747 (15.21) Apple Inc Chief Executive [Tim Cook] has jetted into China ... [Cook] is on his
7748 first business trip to the country ...

7749 In this news article, the family name *Cook* is used as a referring expression; in informal
7750 conversation, it might be more typical to use the given name *Tim*, while more formal
7751 venues, such as *The Economist*, would use the title form *Mr Cook*.

7752 A typical solution for proper noun coreference is to match the syntactic **head words**
7753 of the reference with the referent. In § 10.5.2, we saw that the head word of a phrase can
7754 be identified by applying head percolation rules to the phrasal parse tree; alternatively,
7755 the head can be identified as the root of the dependency subtree covering the name. For
7756 sequences of proper nouns, the head word will be the final token.

7757 There are a number of caveats to the practice of matching head words of proper nouns.

- 7758 • In the European tradition, family names tend to be more specific than given names,
7759 and family names usually come last. However, other traditions have other practices:
7760 for example, in Chinese names, the family name typically comes first; in Japanese,
7761 honorifics come after the name, as in *Nobu-San* (*Mr. Nobu*).
7762 • In organization names, the head word is often not the most informative: for exam-
7763 ple, *Georgia Tech* and *Virginia Tech* are distinguished by the modifiers *Virginia* and
7764 *Georgia*, and not the heads. Similarly, *Lebanon* does not refer to the same entity as
7765 *Southern Lebanon*, necessitating special rules for the specific case of geographical
7766 modifiers (Lee et al., 2011).
7767 • Proper nouns can be nested, as in [*the CEO of [Microsoft]*], resulting in head word
7768 match without coreference.

Despite these difficulties, proper nouns are the easiest category of references to resolve (Stoyanov et al., 2009). In machine learning systems, one solution is to include a range of matching features, including exact match, head match, and string inclusion. In addition to matching features, competitive systems (e.g., Bengtson and Roth, 2008) include large lists, or **gazetteers**, of acronyms (e.g., *the National Basketball Association/NBA*), demonyms (e.g., *the Israelis/Israel*), and other aliases (e.g., *the Georgia Institute of Technology/Georgia Tech*).

15.1.3 Nominals

In coreference resolution, noun phrases that are neither pronouns nor proper nouns are referred to as **nominals**. In the running example (Figure 15.1), nominal references include:

- *the firm (Apple Inc)*
- *the firm's biggest growth market (China)*
- *the country (China)*.

Nominals are especially difficult to resolve (Denis and Baldridge, 2007; Durrett and Klein, 2013), and the examples above suggest why this may be the case: world knowledge is required to identify *Apple Inc* as a *firm*, and *China* as a *growth market*. Other difficult examples include the use of colloquial expressions, such as coreference between *Clinton campaign officials* and *the Clinton camp* (Soon et al., 2001).

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:

$$c_1 = \{Apple\ Inc_{1:2}, the\ firm_{27:28}\} \quad [15.1]$$

$$c_2 = \{Apple\ Inc\ Chief\ Executive\ Tim\ Cook_{1:6}, he_{17}, Cook_{33}, his_{36}\} \quad [15.2]$$

$$c_3 = \{China_{10}, the\ firm\ 's\ biggest\ growth\ market_{27:32}, the\ country_{40:41}\} \quad [15.3]$$

Each row specifies the token spans that mention an entity. (“Singleton” entities, which are mentioned only once (e.g., *talks, government officials*), are excluded.) Equivalently, if given a set of M mentions, $\{m_i\}_{i=1}^M$, each mention i can be assigned to a cluster z_i , where $z_i = z_j$ if i and j are coreferent. The cluster assignments z are invariant under permutation. The unique clustering associated with the assignment z is written $c(z)$.

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 such as pleonastic *it*, 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.

7793 **Mention identification** The task of identifying mention spans for coreference resolution
 7794 is often performed by applying a set of heuristics to the phrase structure parse of each
 7795 sentence. A typical approach is to start with all noun phrases and named entities, and then
 7796 apply filtering rules to remove nested noun phrases with the same head (e.g., [Apple CEO
 7797 [Tim Cook]]), numeric entities (e.g., [100 miles], [97%]), pleonastic *it*, etc (Lee et al., 2013;
 7798 Durrett and Klein, 2013). In general, these deterministic approaches err in favor of recall,
 7799 since the mention clustering component can choose to ignore false positive mentions, but
 7800 cannot recover from false negatives. An alternative is to consider all spans (up to some
 7801 finite length) as candidate mentions, performing mention identification and clustering
 7802 jointly (Daumé III and Marcu, 2005; Lee et al., 2017).

7803 **Mention clustering** The overwhelming majority of research on coreference resolution
 7804 addresses the subtask of mention clustering, and this will be the focus of the remainder
 7805 of this chapter. There are two main sets of approaches, and as usual, they are distin-
 7806 guished by an independence assumption. In *mention-based models*, the scoring function
 7807 for a coreference clustering decomposes over pairs of mentions. These pairwise decisions
 7808 are then aggregated, using a clustering heuristic. Mention-based coreference clustering
 7809 can be treated as a fairly direct application of supervised classification or ranking. How-
 7810 ever, the mention-pair independence assumption can result in incoherent clusters, like
 7811 {*Hillary Clinton* ← *Clinton* ← *Mr Clinton*}, in which the pairwise links score well, but the
 7812 overall result is unsatisfactory. *Entity-based models* address this issue by scoring entities
 7813 holistically. This can make inference more difficult, since the number of possible entity
 7814 groupings is exponential in the number of mentions.

7815 15.2.1 Mention-pair models

7816 In the **mention-pair model**, a binary label $y_{i,j} \in \{0, 1\}$ is assigned to each pair of mentions
 7817 (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
 7818 mention *he* in Figure 15.1 is preceded by five other mentions: (1) *Apple Inc*; (2) *Apple Inc*
 7819 *Chief Executive Tim Cook*; (3) *China*; (4) *talks*; (5) *government officials*. The correct mention
 7820 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,
 7821 such as mention 3 in the example, then $y_{i,j} = 0$ for all i . The same is true for “mentions”
 7822 that do not refer to any entity, such as pleonastic pronouns. If mention j refers to an entity
 7823 that has been mentioned more than once, then $y_{i,j} = 1$ for all $i < j$ that mention the
 7824 referent.

7825 By transforming coreference into a set of binary labeling problems, the mention-pair
 7826 model makes it possible to apply an off-the-shelf binary classifier (Soon et al., 2001). This
 7827 classifier is applied to each mention j independently, searching backwards from j until
 7828 finding an antecedent i which corefers with j with high confidence. After identifying a
 7829 single **antecedent**, the remaining mention pair labels can be computed by transitivity: if
 7830 $y_{i,j} = 1$ and $y_{j,k} = 1$, then $y_{i,k} = 1$.

7831 Since the ground truth annotations give entity chains c but not individual mention-
 7832 pair labels y , an additional heuristic must be employed to convert the labeled data into
 7833 training examples for classification. A typical approach is to generate at most one positive
 7834 labeled instance $y_{a_i, i} = 1$ for mention i , where $a_i = \max\{j : j < i \wedge z_j = z_i\}$ is the index
 7835 of the most recent mention that refers to the same entity as i . We then generate negative
 7836 labeled instances $y_{i,j} = 0$ for all $i > a_j$. In the running example, the most recent antecedent
 7837 of the pronoun *he* is $a_6 = 2$, so the training data would be:

$$y_{2,6} = 1, \quad y_{3,6} = y_{4,6} = y_{5,6} = 0. \quad [15.4]$$

7838 The variable $y_{1,6}$ is not part of the training data, because the first mention appears before
 7839 the true antecedent $a_6 = 2$.

7840 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 ,

$$\hat{a}_i = \operatorname{argmax}_{a \in \{\epsilon, 1, 2, \dots, i-1\}} \psi_M(a, i), \quad [15.5]$$

7841 where $\psi_M(a, i)$ is a score for the mention pair (a, i) . If $a = \epsilon$, then mention i does not refer
 7842 to any previously-introduced entity — it is not **anaphoric**. Mention-ranking is similar to
 7843 the mention-pair model, but all candidates are considered simultaneously, and at most
 7844 a single antecedent is selected. The mention-ranking model explicitly accounts for the
 7845 possibility that mention i is not anaphoric, through the score $\psi_M(\epsilon, i)$. The determination
 7846 of anaphoricity can be made by a special classifier in a preprocessing step, so that non- ϵ
 7847 antecedents are identified only for spans that are determined to be anaphoric (Denis and
 7848 Baldridge, 2008).

7849 As a learning problem, ranking can be trained using the same objectives as in dis-
 7850 criminative classification. For each mention i , we can define a gold antecedent a_i^* , and an
 7851 associated loss, such as the hinge loss, $\ell_i = (1 - \psi_M(a_i^*, i) + \psi_M(\hat{a}, i))_+$ or the negative
 7852 log-likelihood, $\ell_i = -\log p(a_i^* | i; \theta)$. (For more on learning to rank, see § 17.1.1.) But as
 7853 with the mention-pair model, there is a mismatch between the labeled data, which comes
 7854 in the form of mention sets, and the desired supervision, which would indicate the spe-
 7855 cific antecedent of each mention. The antecedent variables $\{a_i\}_{i=1}^M$ relate to the mention
 7856 sets in a many-to-one mapping: each set of antecedents induces a single clustering, but a
 7857 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.
 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):

$$\hat{\mathbf{a}} = \operatorname{argmax}_{\mathbf{a}} \sum_{i=1}^M \psi_M(a_i, i) \quad [15.6]$$

$$\mathbf{a}^* = \operatorname{argmax}_{\mathbf{a} \in \mathcal{A}(c)} \sum_{i=1}^M \psi_M(a_i, i) \quad [15.7]$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \sum_{i=1}^M \frac{\partial L}{\partial \boldsymbol{\theta}} \psi_M(a_i^*, i) - \sum_{i=1}^M \frac{\partial L}{\partial \boldsymbol{\theta}} \psi_M(\hat{a}_i, i) \quad [15.8]$$

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:

$$p(c | \mathbf{m}) = \sum_{\mathbf{a} \in \mathcal{A}(c)} p(\mathbf{a} | \mathbf{m}) = \sum_{\mathbf{a} \in \mathcal{A}(c)} \prod_{i=1}^M p(a_i | i, \mathbf{m}) \quad [15.9]$$

$$p(a_i | i, \mathbf{m}) = \frac{\exp(\psi_M(a_i, i))}{\sum_{a' \in \{\epsilon, 1, 2, \dots, i-1\}} \exp(\psi_M(a', i))}. \quad [15.10]$$

7858 This objective rewards models that assign high scores for all valid antecedent structures.
 7859 In the running example, this would correspond to summing the probabilities of the two
 7860 valid antecedents for *Cook, he* and *Apple Inc Chief Executive Tim Cook*. In one of the exer-
 7861 cises, you will compute the number of valid antecedent structures for a given clustering.

7862 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:

$$m_1 = \text{Hillary Clinton} \quad [15.11]$$

$$m_2 = \text{Clinton} \quad [15.12]$$

$$m_3 = \text{Bill Clinton} \quad [15.13]$$

7863 A mention-pair system might predict $\hat{y}_{1,2} = 1, \hat{y}_{2,3} = 1, \hat{y}_{1,3} = 0$. Similarly, a mention-
 7864 ranking system might choose $\hat{a}_2 = 1$ and $\hat{a}_3 = 2$. Logically, if mentions 1 and 3 are both
 7865 coreferent with mention 2, then all three mentions must refer to the same entity. This
 7866 constraint is known as **transitive closure**.

Transitive closure can be applied *post hoc*, revising the independent mention-pair or mention-ranking decisions. However, there are many possible ways to enforce transitive closure: in the example above, we could set $\hat{y}_{1,3} = 1$, or $\hat{y}_{1,2} = 0$, or $\hat{y}_{2,3} = 0$. For documents with many mentions, there may be many violations of transitive closure, and many possible fixes. Transitive closure can be enforced by always adding edges, so that $\hat{y}_{1,3} = 1$ is preferred (e.g., Soon et al., 2001), but this can result in overclustering, with too many mentions grouped into too few entities.

Mention-pair coreference resolution can be viewed as a constrained optimization problem,

$$\begin{aligned} \max_{\mathbf{y} \in \{0,1\}^M} \quad & \sum_{j=1}^M \sum_{i=1}^j \psi_M(i, j) \times y_{i,j} \\ \text{s.t.} \quad & y_{i,j} + y_{j,k} - 1 \leq y_{i,k}, \quad \forall i < j < k, \end{aligned}$$

with the constraint enforcing transitive closure. This constrained optimization problem is equivalent to graph partitioning with positive and negative edge weights: construct a graph where the nodes are mentions, and the edges are the pairwise scores $\psi_M(i, j)$; the goal is to partition the graph so as to maximize the sum of the edge weights between all nodes within the same partition (McCallum and Wellner, 2004). This problem is NP-hard, motivating approximations such as correlation clustering (Bansal et al., 2004) and **integer linear programming** (Klenner, 2007; Finkel and Manning, 2008, also see § 13.2.2).

15.2.4 Entity-based models

A key weakness of mention-based models is that they treat coreference resolution as a classification or ranking problem, when in fact it is 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 defining a scoring function at the entity level, measuring whether each set of mentions is internally consistent. Coreference resolution can then be viewed as the following optimization,

$$\max_{\mathbf{z}} \quad \sum_{e=1} \psi_E(\{i : z_i = e\}), \tag{15.14}$$

where z_i indicates the entity referenced by mention i , and $\psi_E(\{i : z_i = e\})$ is a scoring function applied to all mentions i that are assigned to entity e .

Entity-based coreference resolution is conceptually similar to the unsupervised clustering problems encountered in chapter 5: the goal is to obtain clusters of mentions that are internally coherent. The number of possible clusterings is the **Bell number**, which is

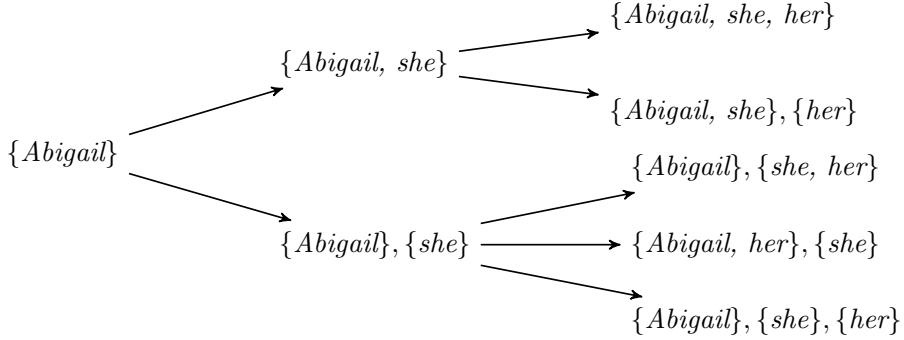


Figure 15.4: The Bell Tree for the sentence *Abigail says she cooks for her*. Which paths are excluded by the syntactic constraints mentioned in § 15.1.1?

defined by the following recurrence (Bell, 1934; Luo et al., 2004),

$$B_n = \sum_{k=0}^{n-1} B_k \binom{n-1}{k} = \frac{1}{e} \sum_{k=0}^{\infty} \frac{k^n}{k!}. \quad [15.15]$$

This recurrence is illustrated by the Bell tree, which is applied to a short coreference problem in Figure 15.4. The Bell number B_n grows exponentially with n , making exhaustive search of the space of clusterings impossible. For this reason, entity-based coreference resolution typically involves incremental search, in which clustering decisions are based on local evidence, in the hope of approximately optimizing the full objective in Equation 15.14. This approach is sometimes called **cluster ranking**, in contrast to mention ranking.

***Generative models of coreference** Contemporary state-of-the-art entity-based methods employ incremental clustering, but another line of research focuses on probabilistic **generative models**, in which the mentions in the document are conditioned on a set of latent entities (Haghghi and Klein, 2007, 2010). An advantage of these methods is that they can be learned from unlabeled data (Poon and Domingos, 2008, e.g.); a disadvantage is that probabilistic inference is required not just for learning, but also for prediction. Furthermore, generative models require independence assumptions that are difficult to apply in coreference resolution, where the diverse and heterogeneous features do not admit an easy decomposition into mutually independent subsets.

15.2.4.1 Incremental cluster ranking

The SMASH method (§ 15.1.1) can be extended to entity-based coreference resolution by building up coreference clusters while moving through the document (Cardie and Wagstaff,

7903 1999). At each mention, the algorithm iterates backwards through possible antecedent
 7904 clusters; but unlike SMASH, a cluster is selected only if *all* members of its cluster are com-
 7905 patible with the current mention. As mentions are added to a cluster, so are their features
 7906 (e.g., gender, number, animacy). In this way, incoherent chains like *{Hillary Clinton, Clinton, Bill Clinton}*
 7907 can be avoided. However, an incorrect assignment early in the document — known as a
 7908 **search error** — might make it impossible to correctly resolve references later on.

7909 More sophisticated search strategies can help to ameliorate the risk of search errors.
 7910 One approach is **beam search** (§ 11.3), in which a set of hypotheses is maintained through-
 7911 out search. Each hypothesis represents a path through the Bell tree (Figure 15.4). Hy-
 7912 potheses are “expanded” either by adding the next mention to an existing cluster, or by
 7913 starting a new cluster. Each expansion receives a score, based on Equation 15.14, and the
 7914 top K hypotheses are kept on the beam as the algorithm moves to the next step.

7915 Incremental cluster ranking can be made more accurate by performing multiple passes
 7916 over the document, applying rules (or “sieves”) with increasing recall and decreasing
 7917 precision at each pass (Lee et al., 2013). In the early passes, coreference links are pro-
 7918 posed only between mentions that are highly likely to corefer (e.g., exact string match
 7919 for full names and nominals). Information can then be shared among these mentions,
 7920 so that when more permissive matching rules are applied later, agreement is preserved
 7921 across the entire cluster. For example, in the case of *{Hillary Clinton, Clinton, she}*, the
 7922 name-matching sieve would link *Clinton* and *Hillary Clinton*, and the pronoun-matching
 7923 sieve would then link *she* to the combined cluster. A deterministic multi-pass system
 7924 won nearly every track of the 2011 CoNLL shared task on coreference resolution (Prad-
 7925 han et al., 2011). Given the dominance of machine learning in virtually all other areas
 7926 of natural language processing — and more than fifteen years of prior work on machine
 7927 learning for coreference — this was a surprising result, even if learning-based methods
 7928 have subsequently regained the upper hand (e.g., Lee et al., 2017, the state-of-the-art at
 7929 the time of this writing).

7930 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 cluster-
 ing 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,

$$\psi_E(\{i : z_i = e\}) = \sum_{i:z_i=e} \boldsymbol{\theta} \cdot \mathbf{f}(i, \{j : j < i \wedge z_j = e\}), \quad [15.16]$$

7931 where the score for each cluster is computed as the sum of scores of linking decisions and
 7932 $\mathbf{f}(i, \emptyset)$ is a set of features for the non-anaphoric mention that initiates the cluster. Using

7933 Figure 15.4 as an example, suppose that the ground truth is,

$$\mathbf{c}^* = \{\text{Abigail}, \text{her}\}, \{\text{she}\}, \quad [15.17]$$

7934 but that with a beam of size one, the learner reaches the hypothesis,

$$\hat{\mathbf{c}} = \{\text{Abigail}, \text{she}\}. \quad [15.18]$$

This hypothesis is incompatible with \mathbf{c}^* , so an update is needed:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{f}(\mathbf{c}^*) - \mathbf{f}(\hat{\mathbf{c}}) \quad [15.19]$$

$$= \boldsymbol{\theta} + (\mathbf{f}(\text{Abigail}, \emptyset) + \mathbf{f}(\text{she}, \emptyset)) - (\mathbf{f}(\text{Abigail}, \emptyset) + \mathbf{f}(\text{she}, \{\text{Abigail}\})) \quad [15.20]$$

$$= \boldsymbol{\theta} + \mathbf{f}(\text{she}, \emptyset) - \mathbf{f}(\text{she}, \{\text{Abigail}\}). \quad [15.21]$$

7935 This style of incremental update can also be applied to a margin loss between the gold
 7936 clustering and the top clustering on the beam. By backpropagating from this loss, it is
 7937 possible to train a neural network, in which the score for each entity is a function of em-
 7938 beddings for the entity mentions (Wiseman et al., 2015).

7939 15.2.4.3 Reinforcement learning

7940 Reinforcement learning is a topic worthy of an entire textbook in its own right (Sutton
 7941 and Barto, 1998),⁵ so this section will provide only a very brief overview, in the context of
 7942 coreference resolution. A stochastic **policy** assigns a probability to each possible **action**,
 7943 conditional on the context. The goal is to learn a policy that achieves a high expected
 7944 reward, or equivalently, a low expected cost.

7945 In incremental cluster ranking, a complete clustering on M mentions can be produced
 7946 by a sequence of M actions, in which the action z_i either merges mention i with an existing
 7947 cluster or begins a new cluster. We can therefore create a stochastic policy using the cluster
 7948 scores (Clark and Manning, 2016),

$$\Pr(z_i = e; \boldsymbol{\theta}) = \frac{\exp \psi_E(i \cup \{j : z_j = e\}; \boldsymbol{\theta})}{\sum_{e'} \exp \psi_E(i \cup \{j : z_j = e'\}; \boldsymbol{\theta})}, \quad [15.22]$$

7949 where $\psi_E(i \cup \{j : z_j = e\}; \boldsymbol{\theta})$ is the score under parameters $\boldsymbol{\theta}$ for assigning mention i to
 7950 cluster e . This score can be an arbitrary function of the mention i , the cluster e and its
 7951 (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).

7952 If a policy assigns probability $p(c(z); \theta)$ to clustering $c(z)$, then its expected loss is,

$$L(\theta) = \sum_{c \in \mathcal{C}(m)} p_\theta(c) \times \ell(c), \quad [15.23]$$

7953 where $\mathcal{C}(m)$ is the set of possible clusterings for mentions m . The loss $\ell(c)$ can be based on
 7954 any arbitrary scoring function, including the complex evaluation metrics used in corefer-
 7955 ence resolution (see § 15.4). This is an advantage for reinforcement learning, which can be
 7956 trained directly on the evaluation metric — unlike traditional supervised learning, which
 7957 requires a loss function that is differentiable and decomposable across individual deci-
 7958 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),

$$\frac{\partial}{\partial \theta} L(\theta) = E_{z \sim \mathcal{Z}(m)} \ell(c(z)) \sum_{i=1}^M \frac{\partial}{\partial \theta} \log p(z_i | z_{1:i-1}, m) \quad [15.24]$$

$$\approx \frac{1}{K} \sum_{k=1}^K \ell(c(z^{(k)})) \sum_{i=1}^M \frac{\partial}{\partial \theta} \log p(z_i^{(k)} | z_{1:i-1}^{(k)}, m) \quad [15.25]$$

$$[15.26]$$

7959 where the action sequence $z^{(k)}$ is sampled from the current policy. Unlike the incremental
 7960 perceptron, an update is not made until the complete action sequence is available.

7961 15.2.4.4 Learning to search

7962 Policy gradient can suffer from high variance: while the average loss over K samples is
 7963 asymptotically equal to the expected reward of a given policy, this estimate may not be
 7964 accurate unless K is very large. This can make it difficult to allocate credit and blame to
 7965 individual actions. In **learning to search**, this problem is addressed through the addition
 7966 of an **oracle** policy, which is known to receive zero or small loss. The oracle policy can be
 7967 used in two ways:

- 7968 • The oracle can be used to generate partial hypotheses that are likely to score well,
 7969 by generating i actions from the initial state. These partial hypotheses are then used
 7970 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  $\frac{\partial}{\partial \theta} L(\theta)$ 

```

- 7971 • The oracle can be used to compute the minimum possible loss from a given state, by
 7972 generating $M - i$ actions from the current state until completion. This is known as
 7973 **roll-out**.

7974 The oracle can be combined with the existing policy during both roll-in and roll-out, sam-
 7975 pling actions from each policy (Daumé III et al., 2009). One approach is to gradually
 7976 decrease the number of actions drawn from the oracle over the course of learning (Ross
 7977 et al., 2011).

7978 In the context of entity-based coreference resolution, Clark and Manning (2016) use
 7979 the learned policy for roll-in and the oracle policy for roll-out. Algorithm 18 shows how
 7980 the gradients on the policy weights are computed in this case. In this application, the
 7981 oracle is “noisy”, because it selects the action that minimizes only the *local* loss — the
 7982 accuracy of the coreference clustering up to mention i — rather than identifying the action
 7983 sequence that will lead to the best final coreference clustering on the entire document.
 7984 When learning from noisy oracles, it can be helpful to mix in actions from the current
 7985 policy with the oracle during roll-out (Chang et al., 2015).

7986 **15.3 Representations for coreference resolution**

7987 Historically, coreference resolution has relied on an array of hand-engineered features to
 7988 capture the linguistic constraints and preferences described in § 15.1 (Soon et al., 2001).
 7989 Later work has documented the utility of large feature sets, including lexical and bilex-
 7990 ical features on mention pairs (Björkelund and Nugues, 2011; Durrett and Klein, 2013).
 7991 The most recent and successful methods replace many (but not all) of these features with
 7992 distributed representations of mentions and entities (Wiseman et al., 2015; Clark and Man-
 7993 ning, 2016; Lee et al., 2017).

7994 **15.3.1 Features**

7995 Coreference features generally rely on a preprocessing pipeline to provide part-of-speech
 7996 tagging and phrase structure parsing. This pipeline makes it possible to design features
 7997 that capture many of the phenomena from § 15.1, and it is also necessary for typical ap-
 7998 proaches to mention identification. However, this pipeline may introduce errors, which
 7999 can propagate to the downstream coreference clustering system. Furthermore, the exis-
 8000 tence of such a pipeline presupposes resources such as treebanks, which do not exist for
 8001 many languages.⁶

8002 **15.3.1.1 Mention features**

8003 Features of individual mentions can help to predict anaphoricity. In systems where men-
 8004 tion detection is performed jointly with coreference resolution, they can also predict whether
 8005 a span of text is likely to be a mention. For mention i , typical features include:

8006 **Mention type** Each span can be identified as a pronoun, name, or nominal, using the
 8007 part-of-speech of the head word of the mention: both the Penn Treebank and Uni-
 8008 versal Dependencies tagsets (§ 8.1.1) include tags for pronouns and proper nouns,
 8009 and all other heads can be marked as nominals (Haghghi and Klein, 2009).

8010 **Mention width** The number of tokens in a mention is a rough predictor of its anaphor-
 8011 icity, with longer mentions being less likely to refer back to previously-defined enti-
 8012 ties.

8013 **Lexical features** The first, last, and head words can help to predict anaphoricity; they are
 8014 also useful in conjunction with features such as mention type and part-of-speech,
 8015 providing a rough measure of agreement (Björkelund and Nugues, 2011). The num-
 8016 ber of lexical features can be very large, so it can be helpful to select only frequently-
 8017 occurring features (Durrett and Klein, 2013).

8018 **Morphosyntactic features** These features include the part-of-speech, number, gender, and
 8019 dependency ancestors.

8020 The features for mention i and candidate antecedent a can conjoined, producing joint
 8021 features that can help to assess the compatibility of the two mentions. For example, Dur-
 8022 rett and Klein (2013) conjoin each feature with the mention types of the anaphora and the

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>

8023 antecedent. Coreference resolution corpora such as ACE and OntoNotes contain docu-
8024 ments from various genres. By conjoining the genre with other features, it is possible to
8025 learn genre-specific feature weights.

8026 15.3.1.2 Mention-pair features

8027 For any pair of mentions i and j , typical features include:

8028 **Distance** The number of intervening tokens, mentions, and sentences can all be used as
8029 distance features. These distances can be computed on the surface text, or on a
8030 transformed representation reflecting the breadth-first tree traversal (Figure 15.3).
8031 Rather than using the distances directly, they are typically binned, creating binary
8032 features.

8033 **String match** A variety of string match features can be employed: exact match, suffix
8034 match, head match, and more complex matching rules that disregard irrelevant
8035 modifiers (Soon et al., 2001).

8036 **Compatibility** Whether the anaphor and antecedent agree with respect to morphosyn-
8037 tactic attributes such as gender, number, and animacy.

8038 **Nesting** If one mention is nested inside another (e.g., *[The President of [France]]*), they
8039 generally cannot corefer.

8040 **Same speaker** For documents with quotations, such as news articles, personal pronouns
8041 can be resolved only by determining the speaker for each mention (Lee et al., 2013).
8042 Coreference is also more likely between mentions from the same speaker.

8043 **Gazetteers** These features fire when the anaphor and candidate antecedent appear in a
8044 gazetteer of acronyms (e.g., *USA/United States*, *GATech/Georgia Tech*), demonymns
8045 (e.g., *Israel/Israeli*), or other aliases (e.g., *Knickerbockers/New York Knicks*).

8046 **Lexical semantics** These features use a lexical resource such as WordNet to determine
8047 whether the head words of the mentions are related through synonymy, antonymy,
8048 and hypernymy (§ 4.2).

8049 **Dependency paths** The dependency path between the anaphor and candidate antecedent
8050 can help to determine whether the pair can corefer, under the government and bind-
8051 ing constraints described in § 15.1.1.

8052 Comprehensive lists of mention-pair features are offered by Bengtson and Roth (2008) and
8053 Rahman and Ng (2011). Neural network approaches use far fewer mention-pair features:
8054 for example, Lee et al. (2017) include only speaker, genre, distance, and mention width
8055 features.

8056 **Semantics** In many cases, coreference seems to require knowledge and semantic in-
 8057 ferences, as in the running example, where we link *China* with a *country* and a *growth*
 8058 *market*. Some of this information can be gleaned from WordNet, which defines a graph
 8059 over **synsets** (see § 4.2). For example, one of the synsets of *China* is an instance of an
 8060 *Asian_nation#1*, which in turn is a hyponym of *country#2*, a synset that includes
 8061 *country*.⁷ Such paths can be used to measure the similarity between concepts (Pedersen
 8062 et al., 2004), and this similarity can be incorporated into coreference resolution as a fea-
 8063 ture (Ponzetto and Strube, 2006). Similar ideas can be applied to knowledge graphs in-
 8064 duced from Wikipedia (Ponzetto and Strube, 2007). But while such approaches improve
 8065 relatively simple classification-based systems, they have proven less useful when added
 8066 to the current generation of techniques.⁸ For example, Durrett and Klein (2013) employ
 8067 a range of semantics-based features — WordNet synonymy and hypernymy relations on
 8068 head words, named entity types (e.g., person, organization), and unsupervised clustering
 8069 over nominal heads — but find that these features give minimal improvement over a
 8070 baseline system using surface features.

8071 15.3.1.3 Entity features

8072 Features for entity-mention coreference can be generated by aggregating mention-pair
 8073 features over all mentions in the candidate entity (Culotta et al., 2007; Rahman and Ng,
 8074 2011). Specifically, for each binary mention-pair feature $f(i, j)$, we compute the following
 8075 entity-mention features for mention i and entity $e = \{j : j < i \wedge z_j = e\}$.

- 8076 • ALL-TRUE: Feature $f(i, j)$ holds for all mentions $j \in e$.
- 8077 • MOST-TRUE: Feature $f(i, j)$ holds for at least half and fewer than all mentions $j \in e$.
- 8078 • MOST-FALSE: Feature $f(i, j)$ holds for at least one and fewer than half of all men-
 8079 tions $j \in e$.
- 8080 • NONE: Feature $f(i, j)$ does not hold for any mention $j \in e$.

8081 For scalar mention-pair features, similar aggregation can be performed by computing the
 8082 minimum, maximum, and median values across all mentions in the cluster. Additional
 8083 entity-mention features include the number of mentions currently clustered in the entity,
 8084 and ALL-X and MOST-X features for each mention type.

8085 15.3.2 Distributed representations of mentions and entities

8086 Recent work has emphasized distributed representations of both mentions and entities.
 8087 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.

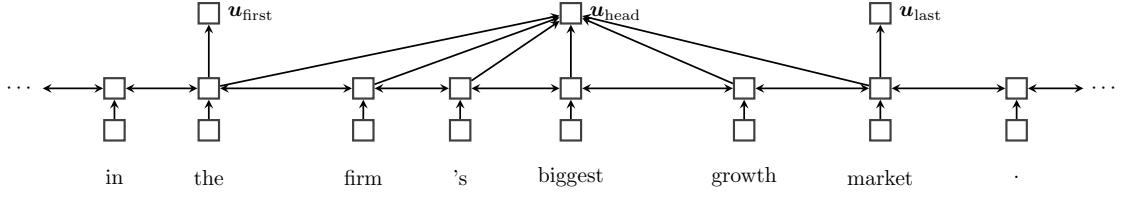


Figure 15.5: A bidirectional recurrent model of mention embeddings. The mention is represented by its first word, its last word, and an estimate of its head word, which is computed from a weighted average (Lee et al., 2017).

8088 mantic compatibility underlying nominal coreference, helping with difficult cases like
 8089 (*Apple, the firm*) and (*China, the firm’s biggest growth market*). Furthermore, a distributed
 8090 representation of entities can be trained to capture semantic features that are added by
 8091 each mention.

8092 15.3.2.1 Mention embeddings

Mention embeddings are based on embeddings of the words in the mention, and the context in which those words appear (Wiseman et al., 2015). A recurrent neural network approach is shown in Figure 15.5. In this approach, a bidirectional long short-term memory (LSTM) is run over the entire document, producing hidden state representations for each word, $\{\mathbf{h}_m\}_{m=1}^M$. For each span (s, t) that is a candidate mention, the distributed representation is the concatenation of four elements: a set of surface features $\mathbf{f}(s, t, \mathbf{w})$, a vector representation of the first word $\mathbf{u}_{\text{first}}^{(s,t)} = \mathbf{h}_s$, the last word $\mathbf{u}_{\text{last}}^{(s,t)} = \mathbf{h}_t$, and the “head” word $\mathbf{u}_{\text{head}}^{(s,t)}$. Rather than identifying the head word from the output of a phrase structure parser, the head can be computed from a neural **attention mechanism**:

$$\tilde{\alpha}_m = \theta_\alpha \cdot \mathbf{h}_m \quad [15.27]$$

$$\mathbf{a}^{(s,t)} = \text{SoftMax}([\tilde{\alpha}_s, \tilde{\alpha}_{s+1}, \dots, \tilde{\alpha}_t]) \quad [15.28]$$

$$\mathbf{u}_{\text{head}}^{(s,t)} = \sum_{m=s}^t a_m^{(s,t)} \mathbf{h}_m. \quad [15.29]$$

8093 The vector $\mathbf{a}^{(s,t)}$ allocates attention across the words in the words in the span (s, t) ; the
 8094 amount of attention is constrained to sum to one by the softmax operation. This elimi-
 8095 nates the need for syntactic parsing to recover the head word, instead learning to identify
 8096 the most important word in the span (Lee et al., 2017). Attention mechanisms were intro-
 8097 duced in neural machine translation (Bahdanau et al., 2014), and are further described in
 8098 chapter 18.

Given a set of mention embeddings, each mention i and candidate antecedent a is scored as,

$$\psi(a, i) = \psi_S(a) + \psi_S(i) + \psi_M(a, i) \quad [15.30]$$

$$\psi_M(a, i) = \text{FeedForward}_S(\mathbf{u}_a) + \text{FeedForward}_S(\mathbf{u}_i) \quad [15.31]$$

$$+ \text{FeedForward}_M([\mathbf{u}_a; \mathbf{u}_i; \mathbf{u}_a \odot \mathbf{u}_i; \mathbf{f}(a, i, \mathbf{w})]), \quad [15.32]$$

where $\text{FeedForward}_S(\mathbf{u}_i)$ is a feedforward neural network applied to the mention representation \mathbf{u}_i . The feature vector $\mathbf{f}(a, i, \mathbf{w})$ describes mentions a and i , including distance, speaker, and genre information. The final term is a feedforward network applied to a vector that vertically concatenates the representations of each mention, their elementwise product $\mathbf{u}_a \odot \mathbf{u}_i$, and the surface features. 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 several 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 pairs, constructing a cluster-pair encoding by **pooling** over the mention-pair encodings for all pairs of mentions within the two clusters.

15.3.2.2 Entity embeddings

In entity-based coreference resolution, each entity should be represented by properties of its mentions. In a distributed setting, we maintain a set of vector entity embeddings, \mathbf{v}_e . Each candidate mention receives an embedding \mathbf{u}_i ; Wiseman et al. (2016) compute this embedding by a single-layer neural network, applied to a vector of surface features. The decision of whether to merge mention i with entity e can then be driven by a feedforward network, $\psi_E(i, e) = \text{Feedforward}([\mathbf{v}_e; \mathbf{u}_i])$. If i is added to entity e , then its representation is updated recurrently, $\mathbf{v}_e \leftarrow f(\mathbf{v}_e, \mathbf{u}_i)$, using a recurrent neural network such as a long short-term memory (LSTM; chapter 6). Alternatively, we can apply a **pooling** operation, such as max-pooling or average-pooling (chapter 3), setting $\mathbf{v}_e \leftarrow \text{Pool}(\mathbf{v}_e, \mathbf{u}_i)$. In either case, the update to the representation of entity e can be thought of as adding new information about the entity from mention i .

8128 15.4 Additional reading

8129 **Historical notes** Ng (2010) surveys coreference resolution through 2010. Early work fo-
8130 cused exclusively on pronoun resolution, with rule-based (Lappin and Leass, 1994) and
8131 probabilistic methods (Ge et al., 1998). The full coreference resolution problem was popu-
8132 larized in a shared task associated with the sixth Message Understanding Conference,
8133 which included coreference annotations for training and test sets of thirty documents
8134 each (Grishman and Sundheim, 1996). An influential early paper was the decision tree
8135 approach of Soon et al. (2001), who introduced mention ranking. A comprehensive list of
8136 surface features for coreference resolution is offered by Bengtson and Roth (2008). Durrett
8137 and Klein (2013) improved on prior work by introducing a large lexicalized feature set;
8138 subsequent work has emphasized neural representations of entities and mentions (Wise-
8139 man et al., 2015).

8140 **Evaluating coreference resolution** The state of coreference evaluation is aggravatingly
8141 complex. Early attempts at simple evaluation metrics were found to under-penalize triv-
8142 ial baselines, such as placing each mention in its own cluster, or grouping all mentions
8143 into a single cluster. Following Denis and Baldridge (2009), the CoNLL 2011 shared task
8144 on coreference (Pradhan et al., 2011) formalized the practice of averaging across three
8145 different metrics: MUC (Vilain et al., 1995), B-CUBED (Bagga and Baldwin, 1998a), and
8146 CEAF (Luo, 2005). Reference implementations of these metrics are available from Pradhan
8147 et al. (2014) at <https://github.com/conll/reference-coreference-scorers>.

8148 Exercises

- 8149 1. Select an article from today’s news, and annotate coreference for the first twenty
8150 noun phrases that appear in the article (include nested noun phrases). That is,
8151 group the noun phrases into entities, where each entity corresponds to a set of noun
8152 phrases. Then specify the mention-pair training data that would result from the first
8153 five noun phrases.
- 8154 2. Using your annotations from the preceding problem, compute the following statis-
8155 tics:
 - 8156 • The number of times new entities are introduced by each of the three types of
8157 referring expressions: pronouns, proper nouns, and nominals. Include “single-
8158 ton” entities that are mentioned only once.
 - 8159 • For each type of referring expression, compute the fraction of mentions that are
8160 anaphoric.

8161 3. Apply a simple heuristic to all pronouns in the article from the previous exercise.
 8162 Specifically, link each pronoun to the closest preceding noun phrase that agrees in
 8163 gender, number, animacy, and person. Compute the following evaluation:

- 8164 • 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.
- 8166 • False positive: a pronoun that is linked to a noun phrase with which it is not
 coreferent. (This includes mistakenly linking singleton or non-referential pro-
 nouns.)
- 8169 • False negative: a pronoun that is not linked to a noun phrase with which it is
 coreferent.

8171 Compute the *F*-MEASURE for your method, and for a trivial baseline in which ev-
 8172 ery mention is its own entity. Are there any additional heuristics that would have
 8173 improved the performance of this method?

- 8174 4. Durrett and Klein (2013) compute the probability of the gold coreference clustering
 by summing over all antecedent structures that are compatible with the clustering.
 Compute the number of antecedent structures for a single entity with K mentions.
- 8177 5. Use the policy gradient algorithm to compute the gradient for the following sce-
 nario, based on the Bell tree in Figure 15.4:
 - 8179 • The gold clustering c^* is $\{Abigail, her\}, \{she\}$.
 - 8180 • Drawing a single sequence of actions ($K = 1$) from the current policy, you
 obtain the following incremental clusterings:

$$\begin{aligned} c(a_1) &= \{Abigail\} \\ c(a_{1:2}) &= \{Abigail, she\} \\ c(a_{1:3}) &= \{Abigail, she\}, \{her\}. \end{aligned}$$

- 8180 • At each mention t , the action space \mathcal{A}_t is to merge the mention with each exist-
 ing cluster, or the empty cluster, with probability,

$$\Pr(\text{Merge}(m_t, c(a_{1:t-1}))) \propto \exp \psi_E(m_t \cup c(a_{1:t-1})), \quad [15.33]$$

8182 where the cluster score $\psi_E(m_t \cup c)$ is defined in Equation 15.16.

8183 Compute the gradient $\frac{\partial}{\partial \theta} L(\theta)$ in terms of the loss $\ell(c(a))$ and the features of each
 8184 (potential) cluster. Explain the differences between the gradient-based update $\theta \leftarrow \theta - \frac{\partial}{\partial \theta} L(\theta)$
 8185 and the incremental perceptron update from this sample example.

8186 Chapter 16

8187 Discourse

8188 Applications of natural language processing often concern multi-sentence documents:
8189 from paragraph-long restaurant reviews, to 500-word newspaper articles, to 500-page
8190 novels. Yet most of the methods that we have discussed thus far are concerned with
8191 individual sentences. This chapter discusses theories and methods for handling multi-
8192 sentence linguistic phenomena, known collectively as **discourse**. There are diverse char-
8193 acterizations of discourse structure, and no single structure is ideal for every computa-
8194 tional application. This chapter covers some of the most well studied discourse repre-
8195 sentations, while highlighting computational models for identifying and exploiting these
8196 structures.

8197 16.1 Segments

8198 A document or conversation can be viewed as a sequence of **segments**, each of which is
8199 **cohesive** in its content and/or function. In Wikipedia biographies, these segments often
8200 pertain to various aspects to the subject's life: early years, major events, impact on others,
8201 and so on. This segmentation is organized around **topics**. Alternatively, scientific research
8202 articles are often organized by **functional themes**: the introduction, a survey of previous
8203 research, experimental setup, and results.

8204 Written texts often mark segments with section headers and related formatting de-
8205 vices. However, such formatting may be too coarse-grained to support applications such
8206 as the retrieval of specific passages of text that are relevant to a query (Hearst, 1997).
8207 Unformatted speech transcripts, such as meetings and lectures, are also an application
8208 scenario for segmentation (Carletta, 2007; Glass et al., 2007; Janin et al., 2003).

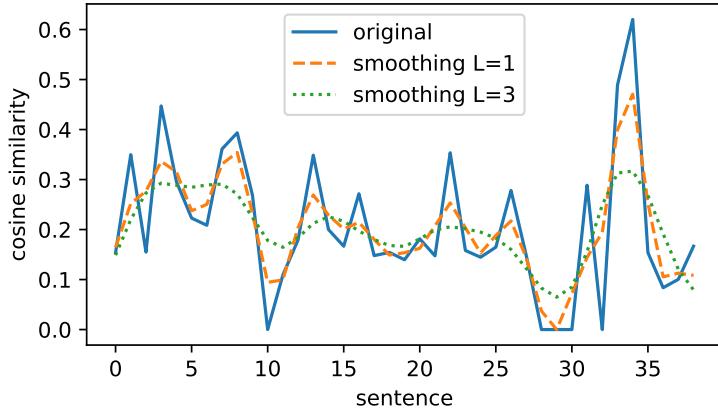


Figure 16.1: Smoothed cosine similarity among adjacent sentences in a news article. Local minima at $m = 10$ and $m = 29$ indicate likely segmentation points.

8209 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,

$$s_m = \frac{\mathbf{x}_m \cdot \mathbf{x}_{m+1}}{\|\mathbf{x}_m\|_2 \times \|\mathbf{x}_{m+1}\|_2} \quad [16.1]$$

$$\bar{s}_m = \sum_{\ell=0}^L k_\ell (s_{m+\ell} + s_{m-\ell}), \quad [16.2]$$

8210 with k_ℓ representing the value of a smoothing kernel of size L . Segmentation points are
 8211 then identified at local minima in the smoothed similarities \bar{s} , since these points indicate
 8212 changes in the overall distribution of words in the text. An example is shown in Figure 16.1.
 8213

8214 Lexical cohesion can also be formulated as a minimum-cut graph segmentation prob-
 8215 lem (Malioutov and Barzilay, 2006) and as a probabilistic model, in which topic segments
 8216 are latent variables (Utiyama and Isahara, 2001; Eisenstein and Barzilay, 2008; Du et al.,

8217 2013).¹ The probabilistic approach can be extended to **hierarchical topic segmentation**,
8218 in which each topic segment is divided into subsegments (Eisenstein, 2009). All of these
8219 approaches are unsupervised. While labeled data can be obtained from well-formatted
8220 texts such as textbooks, such annotations may not generalize to speech transcripts in al-
8221 ternative domains. Supervised methods have been tried in cases where in-domain labeled
8222 data is available, substantially improving performance by learning weights on multiple
8223 types of features (Galley et al., 2003).

8224 16.1.2 Functional segmentation

8225 In some genres, there is a canonical set of communicative *functions*: for example, in sci-
8226 entific research articles, one such function is to communicate the general background for
8227 the article, another is to introduce a new contribution, or to describe the aim of the re-
8228 search (Teufel et al., 1999). A **functional segmentation** divides the document into con-
8229 tiguous segments, sometimes called **rhetorical zones**, in which each sentence has the same
8230 function. Teufel and Moens (2002) train a supervised classifier to identify the functional
8231 of each sentence in a set of scientific research articles, using features that describe the sen-
8232 tence’s position in the text, its similarity to the rest of the article and title, tense and voice of
8233 the main verb, and the functional role of the previous sentence. Functional segmentation
8234 can also be performed without supervision. Noting that some types of Wikipedia arti-
8235 cles have very consistent functional segmentations (e.g., articles about cities or chemical
8236 elements), Chen et al. (2009) introduce an unsupervised model for functional segmenta-
8237 tion, which learns both the language model associated with each function and the typical
8238 patterning of functional segments across the article.

8239 16.2 Entities and reference

8240 Another dimension of discourse relates to which entities are mentioned throughout the
8241 text, and how. Consider the examples in Figure 16.2: Grosz et al. (1995) argue that the first
8242 discourse is more coherent. Do you agree? The examples differ in their choice of **refe-
8243 ring expressions** for the protagonist *John*, and in the syntactic constructions in sentences
8244 (b) and (d). The examples demonstrate the need for theoretical models to explain how
8245 referring expressions are chosen, and where they are placed within sentences. Such mod-
8246 els can then be used to help interpret the overall structure of the discourse, to measure
8247 discourse coherence, and to generate discourses in which referring expressions are used
8248 coherently.

¹There is a rich literature on how latent variable models (such as **latent Dirichlet allocation**) can track 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.

8249 16.2.1 Centering theory

8250 The relationship between discourse and entity reference is most elaborated in **centering**
8251 **theory** (Grosz et al., 1995). According to the theory, every utterance in the discourse is
8252 characterized by a set of entities, known as *centers*.

- 8253 • The **forward-looking centers** are all the entities that are mentioned in an utterance,
8254 $c_f(w_m) = \{e_1, e_2, \dots\}$. The forward-looking centers are partially ordered by their
8255 prominence in the utterance, as measured by phenomena such as syntactic position
8256 (subject, object, other).
- 8257 • The **backward-looking center** $c_b(w_m)$ is the highest-ranked element in the previous
8258 set of forward-looking centers $c_f(w_{m-1})$ that is also mentioned in w_m .

8259 Given these two definitions, centering theory makes the following predictions about the
8260 form and position of referring expressions:

- 8261 1. If a pronoun appears in the utterance w_m , then the backward-looking center $c_b(w_m)$
8262 must also be realized as a pronoun. This rule argues against the use of *it* to refer to
8263 the piano store in Example (16.2d), since the piano store is not mentioned in (16.2c),
8264 and therefore cannot be the backward-looking center of (16.2d).
- 8265 2. Sequences of utterances should retain the same backward-looking center if possible,
8266 and ideally, the backward-looking center should also be the top-ranked element in
8267 the list of forward-looking centers. This rule argues in favor of the preservation of
8268 John as the backward-looking center throughout Example (16.1).

8269 Centering theory unifies aspects of syntax, discourse, and anaphora resolution. However,
8270 it can be difficult to clarify exactly how to rank the elements of each utterance, or even
8271 how to partition a text or dialog into utterances (Poesio et al., 2004).

	Skyler	Walter	danger	a guy	the door
You don't know who you're talking to,	S	-	-	-	-
so let me clue you in.	O	O	-	-	-
I am not in danger, Skyler.	X	S	X	-	-
I am the danger.	-	S	O	-	-
A guy opens his door and gets shot,	-	-	-	S	O
and you think that of me?	S	X	-	-	-
No. I am the one who knocks!	-	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

8290 text (Lapata, 2003), and disentangling multiple conversational threads in an online multi-
 8291 party chat (Elsner and Charniak, 2010).

8292 **16.2.3 *Formal semantics beyond the sentence level**

8293 An alternative view of the role of entities in discourse focuses on formal semantics, and the
 8294 construction of meaning representations for multi-sentence units. Consider the following
 8295 two sentences (from Bird et al., 2009):

- 8296 (16.3) a. Angus owns a dog.
 8297 b. It bit Irene.

8298 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, consider the semantic representations of each individual sentence:

$$\exists x. \text{DOG}(x) \wedge \text{OWN}(\text{Angus}, x) \quad [16.5]$$

$$\text{BITE}(y, \text{Irene}). \quad [16.6]$$

8299 Unifying these two representations into the form of Equation 16.4 requires linking the
 8300 unbound variable y from [16.6] with the quantified variable x in [16.5]. Discourse under-
 8301 standing therefore requires the reader to update a set of assignments, from variables
 8302 to entities. This update would (presumably) link the *dog* in the first sentence of [16.3]
 8303 with the unbound variable y in the second sentence, thereby licensing the conjunction
 8304 in [16.4]. This basic idea is at the root of **dynamic semantics** (Groenendijk and Stokhof,
 8305 1991). **Segmented discourse representation theory** links dynamic semantics with a view
 8306 of discourse that is based on relations between discourse units (Lascarides and Asher,
 8307 2007).

8308 **16.3 Relations**

8309 In dependency grammar, sentences are characterized by a graph (usually a tree) of syntac-
 8310 tic relations between words, such as NSUBJ and DET. A similar idea can be applied at the
 8311 document level, identifying relations between discourse units, such as clauses, sentences,
 8312 or paragraphs. The task of **discourse parsing** involves identifying discourse units and
 8313 the relations that hold between them. These relations can then be applied to tasks such as
 8314 document classification and summarization.

- 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.

8315 16.3.1 Shallow discourse relations

8316 The existence of discourse relations is hinted by **discourse connectives**, such as *however*,
 8317 *moreover*, *meanwhile*, and *if ... then*. These connectives explicitly specify the relationship
 8318 between adjacent units of text: *however* signals a contrastive relationship, *moreover* signals
 8319 that the subsequent text elaborates or strengthens the point that was made immediately
 8320 beforehand, *meanwhile* indicates that two events are contemporaneous, and *if ... then* sets
 8321 up a conditional relationship. Discourse connectives can therefore be viewed as a starting
 8322 point for the analysis of discourse structure.

8323 In **lexicalized tree-adjoining grammar for discourse (D-LTAG)**, each connective an-
 8324 chors a relationship between two units of text (Webber, 2004). This model provides the
 8325 theoretical basis for the **Penn Discourse Treebank (PDTB)**, the largest corpus of discourse
 8326 relations in English (Prasad et al., 2008). It includes a hierarchical inventory of discourse
 8327 relations (shown in Table 16.1), which is created by abstracting the meanings implied by
 8328 the discourse connectives that appear in real texts (Knott, 1996). These relations are then
 8329 annotated on the same corpus of news text used in the Penn Treebank (see § 9.2.2), adding
 8330 the following information:

- 8331 • Each connective is annotated for the discourse relation that it expresses (if any).
- 8332 • For each discourse relation, the two arguments of the relation are specified as ARG1
- 8333 and ARG2, where ARG2 is constrained to be adjacent to the connective. These argu-
- 8334 ments may be sentences, but they may also smaller or larger units of text.
- 8335 • Adjacent sentences are annotated for **implicit discourse relations**, which are not
- 8336 marked by any connective. When a connective could be inserted between a pair of
- 8337 sentence, the annotator supplies it, and also labels its sense (e.g., item 16.5). In some
- 8338 cases, there is no relationship at all between a pair of adjacent sentences; in other
- 8339 cases, the only relation is that the adjacent sentences mention one or more shared
- 8340 entity. These phenomena are annotated as NOREL and ENTREL (entity relation),
- 8341 respectively.

8342 Examples of Penn Discourse Treebank annotations are shown in Figure 16.4. In (16.4),
 8343 the word *therefore* acts as an explicit discourse connective, linking the two adjacent units
 8344 of text. The Treebank annotations also specify the “sense” of each relation, linking the
 8345 connective to a relation in the sense inventory shown in Table 16.1: in (16.4), the relation
 8346 is PRAGMATIC CAUSE:JUSTIFICATION because it relates to the author’s communicative in-
 8347 tentions. The word *therefore* can also signal causes in the external world (e.g., *He was*
 8348 *therefore forced to relinquish his plan*). In **discourse sense classification**, the goal is to de-
 8349 termine which discourse relation, if any, is expressed by each connective. A related task
 8350 is the classification of implicit discourse relations, as in (16.5). In this example, the re-
 8351 lationship between the adjacent sentences could be expressed by the connective *because*,
 8352 indicating a CAUSE:REASON relationship.

8353 16.3.1.1 Classifying explicit discourse relations and their arguments

8354 As suggested by the examples above, many connectives can be used to invoke multiple
 8355 types of discourse relations. Similarly, some connectives have senses that are unrelated
 8356 to discourse: for example, *and* functions as a discourse connective when it links propo-
 8357 sitions, but not when it links noun phrases (Lin et al., 2014). Nonetheless, the senses of
 8358 explicitly-marked discourse relations in the Penn Treebank are relatively easy to classify,
 8359 at least at the coarse-grained level. When classifying the four top-level PDTB relations,
 8360 90% accuracy can be obtained simply by selecting the most common relation for each
 8361 connective (Pitler and Nenkova, 2009). At the more fine-grained levels of the discourse
 8362 relation hierarchy, connectives are more ambiguous. This fact is reflected both in the ac-
 8363 curacy of automatic sense classification (Versley, 2011) and in interannotator agreement,
 8364 which falls to 80% for level-3 discourse relations (Prasad et al., 2008).

8365 A more challenging task for explicitly-marked discourse relations is to identify the
 8366 scope of the arguments. Discourse connectives need not be adjacent to ARG1, as shown
 8367 in (16.6), where ARG1 follows ARG2; furthermore, the arguments need not be contiguous,

- (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).

8368 as shown in (16.7). For these reasons, recovering the arguments of each discourse con-
 8369 nective is a challenging subtask. Because intra-sentential arguments are often syntactic
 8370 constituents (see chapter 10), many approaches train a classifier to predict whether each
 8371 constituent is an appropriate argument for each explicit discourse connective (Wellner
 8372 and Pustejovsky, 2007; Lin et al., 2014, e.g.,).

8373 16.3.1.2 Classifying implicit discourse relations

Implicit discourse relations are considerably more difficult to classify and to annotate.² Most approaches are based on an encoding of each argument, which is then used as input to a non-linear classifier:

$$\mathbf{z}^{(i)} = \text{Encode}(\mathbf{w}^{(i)}) \quad [16.7]$$

$$\mathbf{z}^{(i+1)} = \text{Encode}(\mathbf{w}^{(i+1)}) \quad [16.8]$$

$$\hat{y}_i = \underset{y}{\operatorname{argmax}} \Psi(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}). \quad [16.9]$$

8374 This basic framework can be instantiated in several ways, including both feature-based
 8375 and neural encoders. Several recent approaches are compared in the 2015 and 2016 shared
 8376 tasks at the Conference on Natural Language Learning (Xue et al., 2015, 2016).

²In the dataset for the 2015 shared task on shallow discourse parsing, the interannotator agreement was 91% of explicit discourse relations, and 81% for non-explicit relations, across all levels of detail (Xue et al., 2015).

8377 **Feature-based approaches** Each argument can be encoded into a vector of surface fea-
 8378 tures. The encoding typically includes lexical features (all words, or all content words, or
 8379 a subset of words such as the first three and the main verb), Brown clusters of individ-
 8380 ual words (§ 14.4), and syntactic features such as terminal productions and dependency
 8381 arcs (Pitler et al., 2009; Lin et al., 2009; Rutherford and Xue, 2014). The classification func-
 8382 tion then has two parts. First, it creates a joint feature vector by combining the encodings
 8383 of each argument, typically by computing the cross-product of all features in each encod-
 8384 ing:

$$\mathbf{f}(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i)}) = \{(a \times b \times y) : (\mathbf{z}_a^{(i)} \mathbf{z}_b^{(i+1)})\} \quad [16.10]$$

8385 The size of this feature set grows with the square of the size of the vocabulary, so it can be
 8386 helpful to select a subset of features that are especially useful on the training data (Park
 8387 and Cardie, 2012). After \mathbf{f} is computed, any classifier can be trained to compute the final
 8388 score, $\Psi(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}) = \theta \cdot \mathbf{f}(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)})$.

8389 **Neural network approaches** In neural network architectures, the encoder is learned
 8390 jointly with the classifier as an end-to-end model. Each argument can be encoded using
 8391 a variety of neural architectures (surveyed in § 14.8): recursive (§ 10.6.1; Ji and Eisenstein,
 8392 2015), recurrent (§ 6.3; Ji et al., 2016), and convolutional (§ 3.4; Qin et al., 2017). The clas-
 8393 sification function can then be implemented as a feedforward neural network on the two
 8394 encodings (chapter 3; for examples, see Rutherford et al., 2017; Qin et al., 2017), or as a
 8395 simple bilinear product, $\Psi(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}) = (\mathbf{z}^{(i)})^\top \Theta_y \mathbf{z}^{(i+1)}$ (Ji and Eisenstein, 2015). The
 8396 encoding model can be trained by backpropagation from the classification objective, such
 8397 as the margin loss. Rutherford et al. (2017) show that neural architectures outperform
 8398 feature-based approaches in most settings. While neural approaches require engineering
 8399 the network architecture (e.g., embedding size, number of hidden units in the classifier),
 8400 feature-based approaches also require significant engineering to incorporate linguistic re-
 8401 sources such as Brown clusters and parse trees, and to select a subset of relevant features.

8402 16.3.2 Hierarchical discourse relations

8403 In sentence parsing, adjacent phrases combine into larger constituents, ultimately pro-
 8404 ducing a single constituent for the entire sentence. The resulting tree structure enables
 8405 structured analysis of the sentence, with subtrees that represent syntactically coherent
 8406 chunks of meaning. **Rhetorical Structure Theory (RST)** extends this style of hierarchical
 8407 analysis to the discourse level.

8408 The basic element of RST is the **discourse unit**, which refers to a contiguous span of
 8409 text. **Elementary discourse units** (EDUs) are the atomic elements in this framework, and
 8410 are typically (but not always) clauses.³ Each discourse relation combines two or more

³Details of discourse segmentation can be found in the RST annotation manual (Carlson and Marcu, 2001).

8411 adjacent discourse units into a larger, composite discourse unit; this process ultimately
 8412 unites the entire text into a tree-like structure.⁴

8413 **Nuclearity** In many discourse relations, one argument is primary. For example:

- 8414 (16.8) [LaShawn loves animals]_N
 8415 [She has nine dogs and one pig]_S

8416 In this example, the second sentence provides EVIDENCE for the point made in the first
 8417 sentence. The first sentence is thus the **nucleus** of the discourse relation, and the second
 8418 sentence is the **satellite**. The notion of **nuclearity** is analogous to the head-modifier struc-
 8419 ture of dependency parsing. However, in RST, some relations have multiple nuclei. For
 8420 example, the arguments of the CONTRAST relation are equally important:

- 8421 (16.9) [The clash of ideologies survives this treatment]_N
 8422 [but the nuance and richness of Gorky's individual characters have vanished in the scuffle]_N⁵

8423 Relations that have multiple nuclei are called **coordinating**; relations with a single nu-
 8424 cleus are called **subordinating**. Subordinating relations are constrained to have only two
 8425 arguments, while coordinating relations (such as CONJUNCTION) may have more than
 8426 two.

8427 **RST Relations** Rhetorical structure theory features a large inventory of discourse rela-
 8428 tions, which are divided into two high-level groups: subject matter relations, and pre-
 8429 sentational relations. Presentational relations are organized around the intended beliefs
 8430 of the reader. For example, in the example (16.8), the second discourse unit provides ev-
 8431 idence intended to increase the reader's belief in the proposition expressed by the first
 8432 discourse unit, that *LaShawn loves animals*. In contrast, subject-matter relations are meant
 8433 to communicate additional facts about the propositions contained in the discourse units
 8434 that they relate:

- 8435 (16.10) [the debt plan was rushed to completion]_N
 8436 [in order to be announced at the meeting]_S⁶

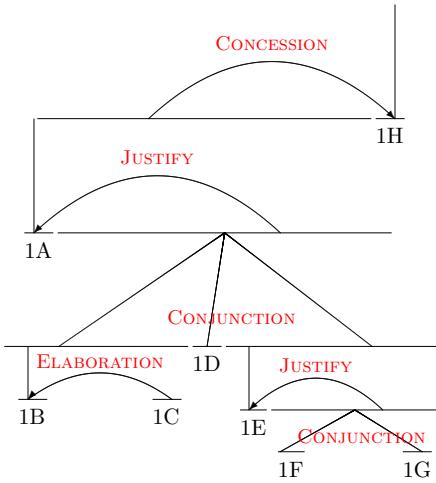
⁴While RST analyses are typically trees, this should be taken as a strong theoretical commitment to the principle that all coherent discourses have a tree structure. Taboada and Mann (2006) write:

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.

⁵from the RST Treebank (Carlson et al., 2002)

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.

8437 In this example, the satellite describes a world state that is realized by the action described
 8438 in the nucleus. This relationship is about the world, and not about the author's commu-
 8439 nicative intentions.

8440 **Example** Figure 16.5 depicts an RST analysis of a paragraph from a movie review. Asym-
 8441 metric (subordinating) relations are depicted with an arrow from the satellite to the nu-
 8442 cleus; symmetric (coordinating) relations are depicted with lines. The elementary dis-
 8443 course units 1F and 1G are combined into a larger discourse unit with the symmetric
 8444 CONJUNCTION relation. The resulting discourse unit is then the satellite in a JUSTIFY
 8445 relation with 1E.

8446 16.3.2.1 Hierarchical discourse parsing

8447 The goal of discourse parsing is to recover a hierarchical structural analysis from a doc-
 8448 ument text, such as the analysis in Figure 16.5. For now, let's assume a segmentation
 8449 of the document into elementary discourse units (EDUs); segmentation algorithms are
 8450 discussed below. After segmentation, discourse parsing can be viewed as a combination
 8451 of two components: the discourse relation classification techniques discussed in § 16.3.1.2,

and algorithms for phrase-structure parsing, such as chart parsing and shift-reduce, which were discussed in chapter 10.

Both chart parsing and shift-reduce require encoding composite discourse units, either in a discrete feature vector or a dense neural representation.⁷ Some discourse parsers rely on the **strong compositionality criterion** (Marcu, 1996), which states that a composite discourse unit can be represented by its nucleus. This criterion is used in feature-based discourse parsing to determine the feature vector for a composite discourse unit (Hernault et al., 2010); it is used in neural approaches to setting the vector encoding for a composite discourse unit equal to the encoding of its nucleus (Ji and Eisenstein, 2014). An alternative neural approach is to learn a composition function over the components of a composite discourse unit (Li et al., 2014), using a recursive neural network (see § 14.8.3).

Bottom-up discourse parsing Assume a segmentation of the text into N elementary discourse units with base representations $\{z^{(i)}\}_{i=1}^N$, and assume a composition function COMPOSE $(z^{(i)}, z^{(j)}, \ell)$, which maps two encodings and a discourse relation ℓ into a new encoding. The composition function can follow the strong compositionality criterion and simply select the encoding of the nucleus, or it can do something more complex. We also need a scoring function $\Psi(z^{(i,k)}, z^{(k,j)}, \ell)$, which computes a scalar score for the (binarized) discourse relation ℓ with left child covering the span $i + 1 : k$, and the right child covering the span $k + 1 : j$. Given these components, we can construct vector representations for each span, and this is the basic idea underlying **compositional vector grammars** (Socher et al., 2013).

These same components can also be used in bottom-up parsing, in a manner that is similar to the CKY algorithm for weighted context-free grammars (see § 10.1): compute the score and best analysis for each possible span of increasing lengths, while storing back-pointers that make it possible to recover the optimal parse of the entire input. However, there is an important distinction from CKY parsing: for each labeled span (i, j, ℓ) , we must use the composition function to construct a representation $z^{(i,j,\ell)}$. This representation is then used to combine the discourse unit spanning $i + 1 : j$ in higher-level discourse relations. The representation $z^{(i,j,\ell)}$ depends on the entire substructure of the unit spanning $i + 1 : j$, and this violates the independence assumption that underlie CKY’s optimality guarantee. Bottom-up parsing with recursively constructed span representations is generally not guaranteed to find the best-scoring discourse parse. This problem is explored in an exercise at the end of the chapter.

Transition-based discourse parsing One drawback of bottom-up parsing is its cubic time complexity in the length of the input. For long documents, transition-based parsing

⁷To 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).

is an appealing alternative. The shift-reduce algorithm can be applied to discourse parsing fairly directly (Sagae, Sagae): the stack stores a set of discourse units and their representations, and each action is chosen by a function of these representations. This function could be a linear product of weights and features, or it could be a neural network applied to encodings of the discourse units. The REDUCE action then performs composition on the two discourse units at the top of the stack, yielding a larger composite discourse unit, which goes on top of the stack. All of the techniques for integrating learning and transition-based parsing, described in § 11.3, are applicable to discourse parsing.

16.3.2.2 Segmenting discourse units

In rhetorical structure theory, elementary discourse units do not cross the sentence boundary, so discourse segmentation can be performed within sentences — as long as the sentence segmentation is given. The segmentation of sentences into elementary discourse units is typically performed using features of the syntactic analysis (Braud et al., 2017). One approach is to train a classifier to determine whether each syntactic constituent is an EDU, using features such as the production, tree structure, and head words (Soricut and Marcu, 2003; Hernault et al., 2010). Another approach is to train a sequence labeling model, such as a conditional random field (Sporleder and Lapata, 2005; Xuan Bach et al., 2012). This is done using the BIO formalism for segmentation by sequence labeling, described in § 8.3.

16.3.3 Argumentation

An alternative view of text-level relational structure focuses on **argumentation** (Stab and Gurevych, 2014b). On this view, each segment (typically a sentence or clause) may support or rebut another segment, creating a graph structure over the text. In the following example (from Peldszus and Stede, 2013), segment S_2 provides argumentative support for the proposition in the segment S_1 :

- (16.11) [We should tear the building down] $_{S1}$
 because it is full of asbestos] $_{S2}$.

Assertions may also support or rebut proposed links between two other assertions, creating a **hypergraph**, which is a generalization of a graph to the case in which edges can join any number of vertices. This can be seen by introducing another sentence into the example:

- (16.12) [In principle it is possible to clean it up] $_{S3}$
 but according to the mayor that is too expensive.] $_{S4}$

8520 S3 acknowledges the validity of S_2 , but undercuts its support of S_1 . This can be repre-
 8521 sented by introducing a hyperedge, $(S_3, S_2, S_1)_{\text{undercut}}$, indicating that S_3 undercuts the
 8522 proposed relationship between S_2 and S_1 . S_4 then undercuts the relevance of S_3 .

8523 **Argumentation mining** is the task of recovering such structures from raw texts. At
 8524 present, annotations of argumentation structure are relatively small: Stab and Gurevych
 8525 (2014a) have annotated a collection of 90 persuasive essays, and Peldszus and Stede (2015)
 8526 have solicited and annotated a set of 112 paragraph-length “microtexts” in German.

8527 16.3.4 Applications of discourse relations

8528 The predominant application of discourse parsing is to select content within a document.
 8529 In rhetorical structure theory, the nucleus is considered the more important element of
 8530 the relation, and is more likely to be part of a summary of the document; it may also
 8531 be more informative for document classification. The D-LTAG theory that underlies the
 8532 Penn Discourse Treebank lacks this notion of nuclearity, but arguments may have varying
 8533 importance, depending on the relation type. For example, the span of text constituting
 8534 ARG1 of an expansion relation is more likely to appear in a summary, while the sentence
 8535 constituting ARG2 of an implicit relation is less likely (Louis et al., 2010). Discourse rela-
 8536 tions may also signal segmentation points in the document structure. Explicit discourse
 8537 markers have been shown to correlate with changes in subjectivity, and identifying such
 8538 change points can improve document-level sentiment classification, by helping the clas-
 8539 sifier to focus on the subjective parts of the text (Trivedi and Eisenstein, 2013; Yang and
 8540 Cardie, 2014).

8541 16.3.4.1 Extractive Summarization

8542 Text **summarization** is the problem of converting a longer text into a shorter one, which
 8543 still conveys the key facts, events, ideas, or sentiments as the original. In **extractive sum-
 8544 marization**, the summary is a subset of the original text; in **abstractive summarization**,
 8545 the summary is produced *de novo*, by paraphrasing the original, or by first encoding it
 8546 into a semantic representation (see § 19.2). In general, extractive summarization meth-
 8547 ods attempt to maximize **coverage**, choosing a subset of the document that covers the
 8548 key concepts mentioned in the document as a whole; typically, coverage is approximated
 8549 by bag-of-words overlap (Nenkova and McKeown, 2012). Coverage-based objectives can
 8550 be supplemented by hierarchical discourse relations, using the principle of nuclearity: in
 8551 any subordinating discourse relation, the nucleus is more critical to the overall meaning
 8552 of the text, and is therefore more important to include in an extractive summary (Marcu,
 8553 1997a).⁸ This insight can be generalized from individual relations using the concept of

⁸Conversely, the arguments of a multi-nuclear relation should either both be included in the summary, or both excluded (Durrett et al., 2016).

8554 **discourse depth** (Hirao et al., 2013): for each elementary discourse unit e , the discourse
 8555 depth d_e is the number of relations in which a discourse unit containing e is the satellite.

Both discourse depth and nuclearity can be incorporated into extractive summarization, using constrained optimization. Let \mathbf{x}_n be a bag-of-words vector representation of elementary discourse unit n , let $y_n \in \{0, 1\}$ indicate whether n is included in the summary, and let d_n be the depth of unit n . Furthermore, let each discourse unit have a “head” h , which is defined recursively: if a discourse unit is produced by a subordinating relation, then its head is the head of the (unique) nucleus; if a discourse unit is produced by a coordinating relation, then its head is the head of the left-most nucleus. For each elementary discourse unit, its parent $\pi(n) \in \{\emptyset, 1, 2, \dots, N\}$ is the head of the smallest discourse unit containing n whose head is not n ; if n is the head of the discourse unit spanning the whole document, then $\pi(n) = \emptyset$. With these definitions in place, discourse-driven summarization can be formalized as (Hirao et al., 2013),

$$\begin{aligned} & \max_{\mathbf{y}=\{0,1\}^N} \sum_{n=1}^N y_n \frac{\Psi(\mathbf{x}_n, \{\mathbf{x}_{1:N}\})}{d_n} \\ & \text{s.t. } \sum_{n=1}^N y_n \left(\sum_{j=1}^V x_{n,j} \right) \leq L \\ & \quad y_{\pi(n)} \geq y_n, \quad \forall n \end{aligned} \tag{16.11}$$

8556 where $\Psi(\mathbf{x}_n, \{\mathbf{x}_{1:N}\})$ measures the coverage of elementary discourse unit n with respect
 8557 to the rest of the document, and $\sum_{j=1}^V x_{n,m}$ is the number of tokens in \mathbf{x}_n . The first con-
 8558 straint ensures that the summary length is less than or equal to some upper bound L . The
 8559 second constraint ensures that no elementary discourse unit is included unless its parent
 8560 is also included. In this way, the discourse structure is used twice: to downweight the
 8561 contributions of elementary discourse units with high depth, and to ensure that the re-
 8562 sulting structure is a subtree of the original discourse parse. The optimization problem in
 8563 16.11 can be solved with **integer linear programming**, described in § 13.2.2.1.⁹

8564 Figure 16.6 shows a **discourse depth tree** for the RST analysis from Figure 16.5, in
 8565 which each elementary discourse is connected to (and below) its parent. The figure also
 8566 shows a valid summary, corresponding to:

8567 (16.13) It could have been a great movie, and I really liked the son of the leader of the
 8568 Samurai. But, other than all that, this movie is nothing more than hidden rip-offs.

⁹Formally, 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. The knapsack problem is NP-hard (Cormen et al., 2009).

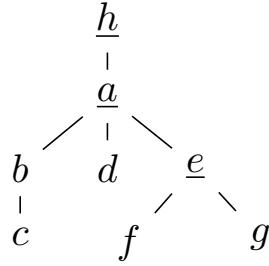


Figure 16.6: A **discourse depth tree** (Hirao et al., 2013) for the discourse parse from Figure 16.5, in which each elementary discourse unit is connected to its parent. The discourse units in one valid summary are underlined.

8569 16.3.4.2 Document classification

8570 Hierarchical discourse structures lend themselves naturally to text classification: in a sub-
 8571 ordinating discourse relation, the nucleus should play a stronger role in the classification
 8572 decision than the satellite. Various implementations of this idea have been proposed.

- 8573 • Focusing on within-sentence discourse relations and lexicon-based classification (see
 8574 § 4.1.2), Voll and Taboada (2007) simply eliminate all words from satellites of each
 8575 discourse relation.
- 8576 • At the document level, elements of each discourse relation argument can be reweighted,
 8577 favoring words in the nucleus, and disfavoring words in the satellite (Heerschap
 8578 et al., 2011; Bhatia et al., 2015). This approach can be applied recursively, computing
 8579 weights across the entire document. The weights can be relation-specific, so that the
 8580 features from the satellites of contrastive relations are discounted or even reversed.
- 8581 • Alternatively, the hierarchical discourse structure can define the structure of a **re-**
 8582 **cursive neural network** (see § 10.6.1). In this network, the representation of each
 8583 discourse unit is computed from its arguments and from a parameter correspond-
 8584 ing to the discourse relation (Ji and Smith, 2017).

8585 Shallow, non-hierarchical discourse relations have also been applied to document clas-
 8586 sification. One approach is to impose a set of constraints on the analyses of individual
 8587 discourse units, so that adjacent units have the same polarity when they are connected
 8588 by a discourse relation indicating agreement, and opposite polarity when connected by a
 8589 contrastive discourse relation, indicating disagreement (Somasundaran et al., 2009; Zirn
 8590 et al., 2011). Yang and Cardie (2014) apply explicitly-marked relations from the Penn
 8591 Discourse Treebank to the problem of sentence-level sentiment polarity classification (see
 8592 § 4.1). They impose the following soft constraints:

- 8593 • When a CONTRAST relation appears between two sentences, those sentences should
8594 have opposite sentiment polarity.
- 8595 • When an EXPANSION or CONTINGENCY relation appears between two sentences,
8596 they should have the same polarity.
- 8597 • When a CONTRAST relation appears *within* a sentence, it should have neutral polar-
8598 ity, since it is likely to express both sentiments.

8599 These discourse-driven constraints are shown to improve performance on two datasets of
8600 product reviews.

8601 16.3.4.3 Coherence

8602 Just as **grammaticality** is the property shared by well-structured sentences, **coherence** is
8603 the property shared by well-structured discourses. One application of discourse process-
8604 ing is to measure (and maximize) the coherence of computer-generated texts like trans-
8605 lations and summaries (Kibble and Power, 2004). Coherence assessment is also used to
8606 evaluate human-generated texts, such as student essays (Miltsakaki and Kukich, 2004;
8607 Burstein et al., 2013, e.g.).

8608 Coherence subsumes a range of disparate phenomena, many of which have been high-
8609 lighted earlier in this chapter: adjacent sentences should be lexically cohesive (Foltz et al.,
8610 1998; Ji et al., 2015; Li and Jurafsky, 2017), and entity references should follow the prin-
8611 ciples of centering theory (Barzilay and Lapata, 2008; Nguyen and Joty, 2017). Discourse
8612 relations also bear on the coherence of a text in a variety of ways:

- 8613 • Hierarchical discourse relations tend to have a “canonical ordering” of the nucleus
8614 and satellite (Mann and Thompson, 1988): for example, in the ELABORATION rela-
8615 tion from rhetorical structure theory, the nucleus always comes first, while in the
8616 JUSTIFICATION relation, the satellite tends to be first (Marcu, 1997b).
- 8617 • Discourse relations should be signaled by connectives that are appropriate to the
8618 semantic or functional relationship between the arguments: for example, a coherent
8619 text would be more likely to use *however* to signal a COMPARISON relation than a
8620 temporal relation (Kibble and Power, 2004).
- 8621 • Discourse relations tend to appear in predictable sequences: for ex-
8622 ample, COMPARISON relations tend to immediately precede CONTINGENCY rela-
8623 tions (Pitler et al., 2008). This observation can be formalized by generalizing the
8624 entity grid model (§ 16.2.2), so that each cell (i, j) provides information about the
8625 role of the discourse argument containing a mention of entity j in sentence i (Lin
8626 et al., 2011). For example, if the first sentence is ARG1 of a comparison relation, then
8627 any entity mentions in the sentence would be labeled COMP.ARG1. This approach

8628 can also be applied to rhetorical structure theory discourse relations (Feng et al.,
8629 2014).

8630 **Datasets** One difficulty with evaluating these methods for measuring discourse coherence
8631 is that human-generated texts usually meet some minimal threshold of coherence.
8632 For this reason, much of the research on measuring coherence has focused on synthetic
8633 data. A typical setting is to permute the sentences of a human-written text, and then de-
8634 termine whether the original sentence ordering scores higher according to the proposed
8635 coherence measure (Barzilay and Lapata, 2008). There are also small datasets of human
8636 evaluations of the coherence of machine summaries: for example, human judgments of
8637 the summaries from the participating systems in the 2003 Document Understanding Con-
8638 ference are available online.¹⁰ Researchers from the Educational Testing Service (an or-
8639 ganization which administers several national exams in the United States) have studied
8640 the relationship between discourse coherence and student essay quality (Burstein et al.,
8641 2003, 2010). A public dataset of essays from second-language learners, with quality anno-
8642 tations, has been made available by researchers at Cambridge University (Yannakoudakis
8643 et al., 2011). At the other extreme, Louis and Nenkova (2013) analyze the structure of
8644 professionally written scientific essays, finding that discourse relation transitions help to
8645 distinguish prize-winning essays from other articles in the same genre.

8646 Additional reading

8647 For a manuscript-length discussion of discourse processing, see Stede (2011). Article-
8648 length surveys are offered by Webber et al. (2012) and Webber and Joshi (2012).

8649 Exercises

- 8650 1.
 - 8651 • Implement the smoothed cosine similarity metric from Equation 16.2, using the
 - 8652 smoothing kernel $\mathbf{k} = [.5, .3, .15, .05]$.
 - 8653 • Download the text of a news article with at least ten paragraphs.
 - 8654 • Compute and plot the smoothed similarity \bar{s} over the length of the article.
 - 8655 • Identify *local minima* in \bar{s} as follows: first find all sentences m such that $\bar{s}_m <$
 - 8656 $\bar{s}_{m \pm 1}$. Then search among these points to find the five sentences with the lowest
 - 8657 \bar{s}_m .
 - 8658 • How often do the five local minima correspond to paragraph boundaries?
 - 8659 – The fraction of local minima that are paragraph boundaries is the **precision-at- k** , where in this case, $k = 5$.

¹⁰<http://homepages.inf.ed.ac.uk/mlap/coherence/>

- 8660 – The fraction of paragraph boundaries which are local minima is the **recall-**
8661 **at- k .**
8662 – Compute precision-at- k and recall-at- k for $k = 3$ and $k = 10$.
- 8663 2. This exercise is to be done in pairs. Each participant selects an article from to-
8664 day's news, and replaces all mentions of individual people with special tokens like
8665 PERSON1, PERSON2, and so on. The other participant should then use the rules
8666 of centering theory to guess each type of referring expression: full name (*Captain*
8667 *Ahab*), partial name (e.g., *Ahab*), nominal (e.g., *the ship's captain*), or pronoun. Check
8668 whether the predictions match the original article, and whether the original article
8669 conforms to the rules of centering theory.
- 8670 3. In § 16.3.2.1, it is noted that bottom-up parsing with compositional representations
8671 of each span is not guaranteed to be optimal. In this exercise, you will construct
8672 a minimal example proving this point. Consider a discourse with four units, with
8673 base representations $\{z^{(i)}\}_{i=1}^4$. Construct a scenario in which the parse selected by
8674 bottom-up parsing is not optimal, and give the precise mathematical conditions that
8675 must hold for this suboptimal parse to be selected. You may ignore the relation
8676 labels ℓ for the purpose of this example.

8677

Part IV

8678

Applications

8679 Chapter 17

8680 Information extraction

8681 Computers offer powerful capabilities for searching and reasoning about structured records
8682 and relational data. Some have even argued that the most important limitation of con-
8683 temporary artificial intelligence is not inference or learning, but simply having too little
8684 knowledge (Lenat et al., 1990). Natural language processing provides an appealing solu-
8685 tion: automatically construct a structured **knowledge base** by reading natural language
8686 text.

8687 Many Wikipedia pages have an “infobox” that provides structured information about
8688 the an entity or event. An example is shown in Figure 17.1a: each row represents one or
8689 more properties of the entity IN THE AEROPLANE OVER THE SEA, a record album. The set
8690 of properties is determined by a predefined **schema**, which applies to all record albums
8691 in Wikipedia. As shown in Figure 17.1b, the values for many of these fields are indicated
8692 directly in the first few sentences of text on the same Wikipedia page.

8693 The task of automatically constructing (or “populating”) an infobox based on the text
8694 is an example of **information extraction**. Much of information extraction can be described
8695 in terms of **entities**, **relations**, and **events**.

8696 • **Entities** are uniquely specified objects in the world, such as people (Jeff Mangum),
8697 places (Athens, Georgia), organizations (Merge Records), and times (February
8698 10, 1998). In chapter 8, we encountered the task of **named entity recognition**,
8699 which labels tokens as parts of entity spans. In information extraction, we must go
8700 further, **linking** each entity **mention** to an element in a **knowledge base**.

- 8701 • **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,...>
```

8702 The set of arguments for an event type is defined by a **schema**. Events often refer to
 8703 time-delimited occurrences: weddings, protests, purchases, terrorist attacks.

8704 Information extraction is similar to predicate-argument semantic parsing tasks, such
 8705 as semantic role labeling (chapter 13): we may think of predicates as corresponding to
 8706 events, and the arguments as defining slots in the event representation. However, the
 8707 goals of information extraction are usually more limited. Rather than accurately pars-
 8708 ing every sentence, information extraction systems often focus on recognizing a few key
 8709 relation or event types, or on the task of identifying all properties of a given entity. Infor-
 8710 mation extraction is often evaluated by the correctness of the resulting knowledge base,
 8711 and not by how many sentences were accurately parsed. Many relations and events will
 8712 be mentioned multiple times in a corpus, but in information extraction, we are usually
 8713 interested in identifying each relation and event only once — thus the goal here is some-
 8714 times described as **macro-reading**, as opposed to **micro-reading**, in which each sentence

(c) Jacob Eisenstein 2018. Work in progress.

must be analyzed correctly. Macro-reading systems are not penalized for ignoring difficult sentences, as long as they can recover the same information from other, easier-to-read sources. However, macro-reading systems must resolve apparent inconsistencies (was the album released on Merge Records or Blue Rose Records?), requiring reasoning across the entire dataset.

In addition to the basic tasks of recognizing entities, relations, and events, accurate information extraction systems must handle negation, and must be able to distinguish statements of fact from hopes, fears, hunches, and hypotheticals. Finally, information extraction is often paired with the problem of **question answering**, which requires accurately parsing a query, and then selecting or generating a textual answer. Question answering systems can be built on knowledge bases that are extracted from large text corpora, or may attempt to identify answers directly from the source texts.

17.1 Entities

The starting point for information extraction is to identify mentions of entities in text. Consider the following text:

(17.4) *The United States Army captured a hill overlooking Atlanta on May 14, 1864.*

For this sentence, we have two goals:

1. **Identify** the spans *United States Army*, *Atlanta*, and *May 14, 1864* as entity mentions. (The hill is not uniquely identified, so it is not a *named* entity.) We may also want to recognize the **named entity types**: organization, location, and date. This is **named entity recognition**, and is described in chapter 8.
2. **Link** these spans to entities in a knowledge base: *U.S. Army*, *Atlanta*, and *1864-May-14*. This task is known as **entity linking**.

The strings to be linked to entities are known as **mentions** — similar to the use of this term in coreference resolution.

- In some formulations of the entity linking task, only named entities are candidates for linking. This is sometimes called **named entity linking** (Ling et al., 2015).
- In other formulations, such as **Wikification** (Milne and Witten, 2008), any string can be a mention.

The set of target entities often corresponds to Wikipedia pages, and Wikipedia is the basis for more comprehensive knowledge bases such as YAGO (Suchanek et al., 2007), DBpedia (Auer et al., 2007), and Freebase (Bollacker et al., 2008). Entity linking may also be

performed in more “closed” settings, where a much smaller list of targets is provided in advance. The system must also determine if a mention does not refer to any entity in the knowledge base, sometimes called a **NIL entity** (McNamee and Dang, 2009).

Returning to (17.4), the three entity mentions may seem unambiguous. But the Wikipedia disambiguation page for the string *Atlanta* says otherwise:¹ there are more than twenty different towns and cities, five United States Navy vessels, a magazine, a television show, a band, and a singer — each prominent enough to have its own Wikipedia page. We now consider how to choose among these dozens of possibilities. In this chapter we will focus on supervised approaches. Unsupervised entity linking is closely related to the problem of **cross-document coreference resolution** (Bagga and Baldwin, 1998b; Singh et al., 2011).

17.1.1 Entity linking by learning to rank

Entity linking is often formulated as a **ranking** problem,

$$\hat{y} = \operatorname{argmax}_{y \in \mathcal{Y}(\mathbf{x})} \psi(y, \mathbf{x}, \mathbf{c}), \quad [17.1]$$

where y is a target entity, \mathbf{x} is a description of the mention, $\mathcal{Y}(\mathbf{x})$ is a set of candidate entities, and \mathbf{c} is a description of the context — such as the other text in the document, or its metadata. The function ψ is a scoring function, which could be a linear model, $\psi(y, \mathbf{x}, \mathbf{c}) = \theta \cdot \mathbf{f}(y, \mathbf{x}, \mathbf{c})$, or a more complex function such as a neural network. In either case, the scoring function can be learned by minimizing a margin-based **ranking loss**,

$$\ell(\hat{y}, y^{(i)}, \mathbf{x}^{(i)}, \mathbf{c}^{(i)}) = \left(\psi(\hat{y}, \mathbf{x}^{(i)}, \mathbf{c}^{(i)}) - \psi(y^{(i)}, \mathbf{x}^{(i)}, \mathbf{c}^{(i)}) + 1 \right)_+, \quad [17.2]$$

where $y^{(i)}$ is the ground truth and $\hat{y} \neq y^{(i)}$ is the predicted target for mention $\mathbf{x}^{(i)}$ in context $\mathbf{c}^{(i)}$ (Joachims, 2002; Dredze et al., 2010).

Candidate identification For computational tractability, it is helpful to restrict the set of candidates, $\mathcal{Y}(\mathbf{x}^{(i)})$. One approach is to use a **name dictionary**, which maps from strings to the entities that they might mention. This mapping is many-to-many: a string such as *Atlanta* can refer to multiple entities, and conversely, an entity such as *Atlanta* can be referenced by multiple strings. A name dictionary can be extracted from Wikipedia, with links between each Wikipedia entity page and the anchor text of all hyperlinks that point to the page (Bunescu and Pasca, 2006; Ratinov et al., 2011). To improve recall, the name dictionary can be augmented by partial and approximate matching (Dredze et al., 2010), but as the set of candidates grows, the risk of false positives (and low precision) increases. For example, the string *Atlanta* is a partial match to *the Atlanta Fed* (a name for the Federal Reserve Bank of Atlanta), and a noisy match (edit distance of one) from *Atalanta* (a heroine in Greek mythology and an Italian soccer team).

¹[https://en.wikipedia.org/wiki/Atlanta_\(disambiguation\)](https://en.wikipedia.org/wiki/Atlanta_(disambiguation)), retrieved November 1, 2017.

8778 **Features** Feature-based approaches to entity ranking rely on three main types of local
 8779 information (Dredze et al., 2010):

- 8780 • The similarity of the mention string to the canonical entity name, as quantified by
 8781 string similarity. This feature would elevate the city Atlanta over the basketball
 8782 team Atlanta Hawks for the string *Atlanta*.
- 8783 • The popularity of the entity, which can be measured by Wikipedia page views or
 8784 PageRank in the Wikipedia link graph. This feature would elevate Atlanta, Georgia
 8785 over the unincorporated community of Atlanta, Ohio.
- 8786 • The entity type, as output by the named entity recognition system. This feature
 8787 would elevate the city of Atlanta over the magazine Atlanta in contexts where
 8788 the mention is tagged as a location.

8789 In addition to these local features, the document context can also help. If *Jamaica* is men-
 8790 tioned in a document about the Caribbean, it is likely to refer to the island nation; in the
 8791 context of New York, it is likely to refer to the neighborhood in Queens; in the context of
 8792 a menu, it might refer to a hibiscus tea beverage. Such hints can be formalized by com-
 8793 puting the similarity between the Wikipedia page describing each candidate entity and
 8794 the context $c^{(i)}$, which may include the bag-of-words representing the document (Dredze
 8795 et al., 2010; Hoffart et al., 2011) or a smaller window of text around the mention (Ratinov
 8796 et al., 2011). Contextual similarity can be modeled using the cosine similarity of bag-of-
 8797 words vectors, typically weighted using **inverse document frequency** to emphasize rare
 8798 words.²

8799 **Neural entity linking** An alternative approach to entity ranking is to compute the score
 8800 for each entity candidate using distributed vector representations of the entities, men-
 8801 tions, and context. For example, for the task of entity linking in Twitter, Yang et al. (2016)
 8802 employ the bilinear scoring function,

$$\psi(y, \mathbf{x}, \mathbf{c}) = \mathbf{v}_y^\top \mathbf{W}^{(y,x)} \mathbf{x} + \mathbf{v}_y^\top \mathbf{W}^{(y,c)} \mathbf{c}, \quad [17.3]$$

8803 with $\mathbf{v}_y \in \mathbb{R}^{K_y}$ as the vector embedding of entity y , $\mathbf{x} \in \mathbb{R}^{K_x}$ as the embedding of the
 8804 mention, $\mathbf{c} \in \mathbb{R}^{K_c}$ as the embedding of the context, and the matrices $\mathbf{W}^{(y,x)}$ and $\mathbf{W}^{(y,c)}$
 8805 as parameters that score the compatibility of each entity with respect to the mention and
 8806 context. Each of the vector embeddings can be learned from an end-to-end objective, or
 8807 pre-trained on unlabeled data.

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).

- 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 on the text of wikipedia, with hyperlinks substituted for the anchor text.³
- The embedding of the mention x can be computed by averaging the embeddings of the words in the mention (Yang et al., 2016), or by one of the compositional techniques described in § 14.8.
- The embedding of the context c can be represented by a low-dimensional vector. In an auto-encoding framework, this vector is the result of noisy compression, so that it is possible to approximately reconstruct the original document (Vincent et al., 2010; Kingma and Welling, 2014). He et al. (2013) apply this idea to entity linking. The vector c can also be obtained by convolution on the embeddings of words in the document (Sun et al., 2015), or by examining metadata such as the author’s social network (Yang et al., 2016).

The remaining parameters $\mathbf{W}^{(y,x)}$ and $\mathbf{W}^{(y,c)}$ can be trained by backpropagation from the margin loss in Equation 17.2.

17.1.2 Collective entity linking

Consider the following lists:

- (17.5) California, Oregon, Washington
- (17.6) Baltimore, Washington, Philadelphia
- (17.7) Washington, Adams, Jefferson

In each case, the term *Washington* refers to a different entity, and this reference is strongly suggested by the other entries on the list. In the last list, all three names are ambiguous in isolation — there are dozens of other *Adams* and *Jefferson* entities in Wikipedia. But a preference for coherence motivates **collectively** linking these references to the first three U.S. presidents.

A general approach to collective entity linking is to introduce a compatibility score $\psi_c(\mathbf{y})$. Collective entity linking is then performed by optimizing the global objective,

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathbb{Y}(\mathbf{x})}{\operatorname{argmax}} \psi_c(\mathbf{y}) + \sum_{i=1}^N \psi_\ell(y^{(i)}, \mathbf{x}^{(i)}, \mathbf{c}^{(i)}), \quad [17.4]$$

where $\mathbb{Y}(\mathbf{x})$ is the set of all possible collective entity assignments for the mentions in \mathbf{x} , and ψ_ℓ is the local scoring function for each entity i . For simplicity, the compatibility func-

³Pre-trained entity embeddings can be downloaded from <https://code.google.com/archive/p/word2vec/>.

8837 tion is typically decomposed into a sum of pairwise scores, $\psi_c(\mathbf{y}) = \sum_{i=1}^N \sum_{j \neq i}^N \psi_c(y^{(i)}, y^{(j)})$.
 8838 The compatibility can then be computed in a number of different ways:

- 8839 • Wikipedia defines high-level categories for entities (e.g., *living people*, *Presidents of*
 8840 *the United States*, *States of the United States*), and ψ_c can reward entity pairs for the
 8841 number of categories that they have in common (Cucerzan, 2007).
- 8842 • Compatibility can be measured by the number of incoming hyperlinks shared by
 8843 the Wikipedia pages for the two entities (Milne and Witten, 2008).
- 8844 • In a neural architecture, the compatibility of two entities can be set equal to the inner
 8845 product of their embeddings, $\psi_c(y^{(i)}, y^{(j)}) = \mathbf{v}_{y^{(i)}} \cdot \mathbf{v}_{y^{(j)}}$.
- 8846 • A non-pairwise compatibility score can be defined using a type of latent variable
 8847 model known as a **probabilistic topic model** (Blei et al., 2003; Blei, 2012). In this
 8848 framework, each latent topic is a probability distribution over entities, and each
 8849 document has a probability distribution over topics. Each entity helps to determine
 8850 the document's distribution over topics, and in turn these topics help to resolve am-
 8851 biguous entity mentions (Newman et al., 2006). Inference can be performed using
 8852 the techniques described in chapter 5.

8853 Unfortunately, collective entity linking is **NP-hard** even for pairwise compatibility func-
 8854 tions, so it almost certainly cannot be solved efficiently. Various approximate inference
 8855 techniques have been proposed, including **integer linear programming** (Cheng and Roth,
 8856 2013), **Gibbs sampling** (Han and Sun, 2012), and graph-based algorithms (Hoffart et al.,
 8857 2011; Han et al., 2011).

8858 17.1.3 *Pairwise ranking loss functions

8859 The loss function defined in Equation 17.2 considers only the highest-scoring prediction
 8860 \hat{y} , but in fact, the true entity $y^{(i)}$ should outscore *all* other entities. A loss function based on
 8861 this idea would give a gradient against the features or representations of several entities,
 8862 not just the top-scoring prediction. Usunier et al. (2009) define a general ranking error
 8863 function,

$$L_{\text{rank}}(k) = \sum_{j=1}^k \alpha_j, \quad \text{with } \alpha_1 \geq \alpha_2 \geq \dots \geq 0, \quad [17.5]$$

8864 where k is equal to the number of labels ranked higher than the correct label $y^{(i)}$. This
 8865 function defines a class of ranking errors: if $\alpha_j = 1$ for all j , then the ranking error is
 8866 equal to the rank of the correct entity; if $\alpha_1 = 1$ and $\alpha_{j>1} = 0$, then the ranking error is
 8867 one whenever the correct entity is not ranked first; if α_j decreases smoothly with j , as in
 8868 $\alpha_j = \frac{1}{j}$, then the error is between these two extremes.

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

```

This ranking error will 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,

$$r(y^{(i)}, \mathbf{x}^{(i)}) \triangleq \sum_{y \in \mathcal{Y}(\mathbf{x}^{(i)}) \setminus y^{(i)}} \delta \left(1 + \psi(y, \mathbf{x}^{(i)}) \geq \psi(y^{(i)}, \mathbf{x}^{(i)}) \right), \quad [17.6]$$

where $\delta(\cdot)$ is a delta function, and $\mathcal{Y}(\mathbf{x}^{(i)}) \setminus y^{(i)}$ is the set of all entity candidates minus the true entity $y^{(i)}$. The margin-augmented rank is the rank of the true entity, after augmenting every other candidate with a margin of one, under the current scoring function ψ . (The context c is omitted for clarity, and can be considered part of \mathbf{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,

$$\ell(y^{(i)}, \mathbf{x}^{(i)}) = \frac{L_{\text{rank}}(r(y^{(i)}, \mathbf{x}^{(i)}))}{r(y^{(i)}, \mathbf{x}^{(i)})} \sum_{y \in \mathcal{Y}(\mathbf{x}) \setminus y^{(i)}} \left(\psi(y, \mathbf{x}^{(i)}) - \psi(y^{(i)}, \mathbf{x}^{(i)}) + 1 \right)_+, \quad [17.7]$$

The sum in Equation 17.7 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 17.7 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

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)

8881 such a violation yields an approximation of the margin-augmented rank of the true en-
 8882 tity, $r(y^{(i)}, \mathbf{x}^{(i)}) \approx \left\lfloor \frac{|\mathcal{Y}(\mathbf{x})|}{N} \right\rfloor$. If a violation is found immediately, $N = 1$, the correct entity
 8883 probably ranks below many others, $r \approx |\mathcal{Y}(\mathbf{x})|$. If many samples are required before a
 8884 violation is found, $N \rightarrow |\mathcal{Y}(\mathbf{x})|$, then the correct entity is probably highly ranked, $r \rightarrow 1$.
 8885 A computational advantage of WARP is that it is not necessary to find the highest-scoring
 8886 label, which can impose a non-trivial computational cost when $\mathcal{Y}(\mathbf{x}^{(i)})$ is large. Note the
 8887 similarity to the **negative sampling** objective in WORD2VEC (chapter 14).

8888 17.2 Relations

8889 Consider the following example:

8890 (17.8) George Bush traveled to France on Thursday for a summit.

8891 This sentence introduces a relation between the entities referenced by *George Bush* and
 8892 *France*. In the Automatic Content Extraction (ACE) ontology (Linguistic Data Consortium,
 8893 2005), the type of this relation is PHYSICAL, and the subtype is LOCATED. This relation
 8894 would be written,

$$\text{PHYSICAL.LOCATED}(\text{George Bush}, \text{France}). \quad [17.8]$$

8895 Relations take exactly two arguments, and the order of the arguments matters: the con-
 8896 verse relation would imply that *France* is inside of *George Bush*, which is completely
 8897 different.

8898 In the ACE datasets, relations are annotated between entity mentions, as in the exam-
 8899 ple above. Relations can also hold between nominals, as in the following example from
 8900 the SemEval-2010 shared task (Hendrickx et al., 2009):

8901 (17.9) The cup contained tea from dried ginseng.

8902 This sentence describes a relation of type ENTITY-ORIGIN between *tea* and *ginseng*. Nom-
 8903 inal relation extraction is closely related to **semantic role labeling** (chapter 13). The key
 8904 difference is that relation extraction is restricted to a relatively small number of relation
 8905 types; for example, Table 17.1 shows the ten relation types from SemEval-2010.

8906 **17.2.1 Pattern-based relation extraction**

8907 Early work on relation extraction focused on hand-crafted patterns (Hearst, 1992). For
 8908 example, the appositive *Starbuck, a native of Nantucket* signals the relation ENTITY-ORIGIN
 8909 between *Starbuck* and *Nantucket*. This pattern can be written as,

$$\text{PERSON}, \text{a native of LOCATION} \Rightarrow \text{ENTITY-ORIGIN}(\text{PERSON}, \text{LOCATION}). \quad [17.9]$$

8910 This pattern will be “triggered” whenever the literal string *, a native of* occurs between an
 8911 entity of type PERSON and an entity of type LOCATION. Such patterns can be generalized
 8912 beyond literal matches using techniques such as lemmatization, which would enable the
 8913 words (*buy, buys, buying*) to trigger the same patterns (see § 4.3.1.2). A more aggressive
 8914 strategy would be to group all words in a WordNet synset (§ 4.2), grouping words like
 8915 *buy* and *purchase*.

8916 Relation extraction patterns can be implemented in finite-state automata (§ 9.1). If
 8917 the named entity recognizer is also a finite-state machine, then the systems can be com-
 8918 bined by finite-state transduction (Hobbs et al., 1997). This makes it possible to propagate
 8919 uncertainty through the finite-state cascade. Suppose the entity recognizer hypothesizes
 8920 that *Starbuck* refers to either a PERSON or a LOCATION; in the composed transducer, the
 8921 relation extractor would be free to select the PERSON annotation when it appears in the
 8922 context of an appropriate pattern.

8923 **17.2.2 Relation extraction as a classification task**

8924 Relation extraction can be formulated as a classification problem,

$$\hat{r}_{(i,j),(m,n)} = \underset{r \in \mathcal{R}}{\operatorname{argmax}} \psi(r, (i, j), (m, n), \mathbf{w}), \quad [17.10]$$

8925 where $r \in \mathcal{R}$ is a relation type (possibly NIL), $\mathbf{w}_{i:j}$ is the span of the first argument, and
 8926 $\mathbf{w}_{m:n}$ is the span of the second argument. The argument $\mathbf{w}_{m:n}$ may appear before or after
 8927 $\mathbf{w}_{i:j}$ in the text, or they may overlap; we stipulate only that $\mathbf{w}_{i:j}$ is the first argument of
 8928 the relation. We now consider three alternatives for computing the scoring function ψ .

8929 17.2.2.1 Feature-based classification

8930 To build a feature-based classifier, we define the scoring function as,

$$\psi(r, (i, j), (m, n), \mathbf{w}) = \boldsymbol{\theta} \cdot \mathbf{f}(r, (i, j), (m, n), \mathbf{w}), \quad [17.11]$$

8931 with $\boldsymbol{\theta}$ representing a vector of weights, and $\mathbf{f}(\cdot)$ a vector of features. The pattern-based
8932 methods described in § 17.2.1 suggest several features:

- 8933 • Local features of $w_{i:j}$ and $w_{m:n}$, including: the strings themselves; whether they are
8934 recognized as entities, and if so, which type; whether the strings are present in a
8935 **gazetteer** of entity names; each string's syntactic **head** (§ 9.2.2).
- 8936 • Features of the span between the two arguments, $w_{j:m}$ or $w_{n:i}$ (depending on which
8937 argument appears first): the length of the span; the specific words that appear in the
8938 span, either as a literal sequence or a bag-of-words; the wordnet synsets (§ 4.2) that
8939 appear in the span between the arguments.
- 8940 • Features of the syntactic relationship between the two arguments, typically the **de-**
8941 **pendency path** between the arguments (§ 13.2.1). Example dependency paths are
8942 shown in Table 17.2.

8943 17.2.2.2 Kernels

8944 Suppose that the first line of Table 17.2 is a labeled example, and the remaining lines are
8945 instances to be classified. A feature-based approach would have to decompose the depen-
8946 dency paths into features that capture individual edges, with or without their labels, and
8947 then learn weights for each of these features: for example, the second line contains identi-
8948 cal dependencies, but different arguments; the third line contains a different inflection of
8949 the word *travel*; the fourth and fifth lines each contain an additional edge on the depen-
8950 dency path; and the sixth example uses an entirely different path. Rather than attempting
8951 to create local features that capture all of the ways in which these dependencies paths
8952 are similar and different, we can instead define a similarity function κ , which computes a
8953 score for any pair of instances, $\kappa : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_+$. The score for any pair of instances (i, j)
8954 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
8955 function κ obeys a few key properties it is called a **kernel function**.⁴

Given a valid kernel function, we can build a non-linear classifier without explicitly defining a feature vector. For a binary classification problem $y \in \{-1, 1\}$, we have the

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).

1. <i>George Bush traveled to France</i>	<i>George Bush</i> \leftarrow <i>traveled</i> \rightarrow <i>France</i> NSUBJ OBL
2. <i>Ahab traveled to Nantucket</i>	<i>Ahab</i> \leftarrow <i>traveled</i> \rightarrow <i>Nantucket</i> NSUBJ OBL
3. <i>George Bush will travel to France</i>	<i>George Bush</i> \leftarrow <i>travel</i> \rightarrow <i>France</i> NSUBJ OBL
4. <i>George Bush wants to travel to France</i>	<i>George Bush</i> \leftarrow <i>wants</i> \rightarrow <i>travel</i> \rightarrow <i>France</i> NSUBJ XCOMP OBL
5. <i>Ahab traveled to a city in France</i>	<i>Ahab</i> \leftarrow <i>traveled</i> \rightarrow <i>city</i> \rightarrow <i>France</i> NSUBJ OBL NMOD
6. <i>We await Ahab's visit to France</i>	<i>Ahab</i> \leftarrow <i>visit</i> \rightarrow <i>France</i> NMOD:POSS NMOD

Table 17.2: Candidates instances for the PHYSICAL.LOCATED relation, and their dependency paths

decision function,

$$\hat{y} = \text{Sign}(b + \sum_{i=1}^N y^{(i)} \alpha^{(i)} \kappa(\mathbf{x}^{(i)}, \mathbf{x})) \quad [17.12]$$

8956 where b and $\{\alpha^{(i)}\}_{i=1}^N$ are parameters that must be learned from the training set, under
 8957 the constraint $\forall_i, \alpha^{(i)} \geq 0$. Intuitively, each α_i specifies the importance of the instance $\mathbf{x}^{(i)}$
 8958 towards the classification rule. Kernel-based classification can be viewed as a weighted
 8959 form of the **nearest-neighbor** classifier (Hastie et al., 2009), in which test instances are
 8960 assigned the most common label among their near neighbors in the training set. This
 8961 results in a non-linear classification boundary. The parameters are typically learned from
 8962 a margin-based objective (see § 2.3), leading to the **kernel support vector machine**. To
 8963 generalize to multi-class classification, we can train separate binary classifiers for each
 8964 label (sometimes called **one-versus-all**), or train binary classifiers for each pair of possible
 8965 labels (**one-versus-one**).

8966 Dependency kernels are particularly effective for relation extraction, due to their abil-
 8967 ity to capture syntactic properties of the path between the two candidate arguments. One
 8968 class of dependency tree kernels is defined recursively, with the score for a pair of trees
 8969 equal to the similarity of the root nodes and the sum of similarities of matched pairs of
 8970 child subtrees (Zelenko et al., 2003; Culotta and Sorensen, 2004). Alternatively, Bunescu
 8971 and Mooney (2005) define a kernel function over sequences of unlabeled dependency
 8972 edges, in which the score is computed as a product of scores for each pair of words in the
 8973 sequence: identical words receive a high score, words that share a synset or part-of-speech
 8974 receive a small non-zero score (e.g., *travel* / *visit*), and unrelated words receive a score of
 8975 zero.

8976 **17.2.2.3 Neural relation extraction**

8977 **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
 8978 matrix of word embeddings \mathbf{X} , where $\mathbf{x}_m \in \mathbb{R}^K$ is the embedding of w_m . Now, suppose
 8979 the candidate arguments appear at positions a_1 and a_2 ; then for each word in the sen-
 8980 tence, its position with respect to each argument is $m - a_1$ and $m - a_2$. (Following Zeng
 8981 et al. (2014), we consider a restricted version of the relation extraction task in which the
 8982 arguments are single tokens.) We can augment the word embeddings by estimating em-
 8983 beddings for these positional offsets, $\mathbf{x}_{m-a_1}^{(p)}$ and $\mathbf{x}_{m-a_2}^{(p)}$. The complete base representation
 8984 of the sentence is,
 8985

$$\mathbf{X}(a_1, a_2) = \begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_M \\ \mathbf{x}_{1-a_1}^{(p)} & \mathbf{x}_{2-a_1}^{(p)} & \cdots & \mathbf{x}_{M-a_1}^{(p)} \\ \mathbf{x}_{1-a_2}^{(p)} & \mathbf{x}_{2-a_2}^{(p)} & \cdots & \mathbf{x}_{M-a_2}^{(p)} \end{pmatrix}, \quad [17.13]$$

8986 where each column is a vertical concatenation of a word embedding, represented by the
 8987 column vector \mathbf{x}_m , and two positional embeddings, specifying the position with respect
 8988 to a_1 and a_2 . The matrix $\mathbf{X}(a_1, a_2)$ is then taken as input to a convolutional layer, and
 8989 max-pooling is applied to obtain a vector. The final scoring function is then,

$$\psi(r, i, j, \mathbf{X}) = \theta_r \cdot \text{MaxPool}(\text{ConvNet}(\mathbf{X}(i, j); \phi)), \quad [17.14]$$

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,

$$\hat{r} = \underset{r}{\operatorname{argmax}} \psi(r, i, j, \mathbf{X}) \quad [17.15]$$

$$\ell = (1 + \psi(\hat{r}, i, j, \mathbf{X}) - \psi(r, i, j, \mathbf{X}))_+. \quad [17.16]$$

8990 **Recurrent neural networks** have also been applied to relation extraction, using a net-
 8991 work such as an bidirectional LSTM to encode the words or dependency path between
 8992 the two arguments. Xu et al. (2015) segment each dependency path into left and right
 8993 subpaths: the path,

8994 (17.10) $George \ Bush \xleftarrow[\text{NSUBJ}]{} wants \xrightarrow[\text{XCOMP}]{} travel \xrightarrow[\text{OBL}]{} France$

8995 is segmented into the subpaths,

8996 (17.11) $George \ Bush \xleftarrow[\text{NSUBJ}]{} wants$

8997 (17.12) $wants \xrightarrow[\text{XCOMP}]{} travel \xrightarrow[\text{OBL}]{} France.$

They 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 $\overleftarrow{\mathbf{h}}_m^{(c)}$ indicate the associated recurrent state in the left subtree. Then the complete model is specified as follows,

$$\mathbf{h}_{s(m)}^{(c)} = \text{RNN}(\mathbf{x}_{s(m)}^{(c)}, \mathbf{h}_m^{(c)}) \quad [17.17]$$

$$\mathbf{z}^{(c)} = \text{MaxPool}\left(\overleftarrow{\mathbf{h}}_i^{(c)}, \overleftarrow{\mathbf{h}}_{s(i)}^{(c)}, \dots, \overleftarrow{\mathbf{h}}_{\text{root}}^{(c)}, \overrightarrow{\mathbf{h}}_j^{(c)}, \overrightarrow{\mathbf{h}}_{s(j)}^{(c)}, \dots, \overrightarrow{\mathbf{h}}_{\text{root}}^{(c)}\right) \quad [17.18]$$

$$\psi(r, i, j) = \theta \cdot [\mathbf{z}^{(\text{word})}; \mathbf{z}^{(\text{POS})}; \mathbf{z}^{(\text{dependency})}; \mathbf{z}^{(\text{hypernym})}]. \quad [17.19]$$

8998 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
 8999 \mathbf{z} . Xu et al. (2015) pass the score ψ through a **softmax** layer to obtain a probability
 9000 $p(r | i, j, \mathbf{w})$, and train the model by regularized **cross-entropy**. Miwa and Bansal (2016)
 9001 show that a related model can solve the more challenging “end-to-end” relation extraction
 9002 task, in which the model must simultaneously detect entities and then extract their
 9003 relations.
 9004

9005 17.2.3 Knowledge base population

9006 In many applications, what matters is not what fraction of sentences are analyzed cor-
 9007 rectly, but how much accurate knowledge can be extracted. **Knowledge base population**
 9008 (**KBP**) refers to the task of filling in Wikipedia-style infoboxes, as shown in Figure 17.1a.
 9009 Knowledge base population can be decomposed into two subtasks: **entity linking** (de-
 9010 scribed in § 17.1), and **slot filling** (Ji and Grishman, 2011). Slot filling has two key differ-
 9011 ences from the formulation of relation extraction presented above:

- 9012 • The relations hold between entities, rather than spans of text
- 9013 • Performance is evaluated at the *type level* (on entity pairs), rather than on the *token*
 9014 *level* (on individual sentences).

9015 From a practical standpoint, there are three other important differences between slot
 9016 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. Aggregating this evidence to predict a single relation type (or set of relations) is a challenge for slot filling systems.
- 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 relatively complex: 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:

- (17.13) Elected mayor of Atlanta in 1973, **Maynard Jackson** was the first African American to serve as mayor of a major southern city.
- (17.14) Atlanta’s airport will be renamed to honor **Maynard Jackson**, the city’s first Black mayor .
- (17.15) Born in Dallas, Texas in 1938, **Maynard Holbrook Jackson, Jr.** moved to Atlanta when he was 8.
- (17.16) **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.

One approach is to run a single-document relation extraction system (using the techniques described in § 17.2.2), and then aggregate the results (Li et al., 2011). Relations

9051 that are detected with high confidence in multiple documents are more likely to be valid,
 9052 motivating the heuristic,

$$\psi(r, e_1, e_2) = \sum_{i=1}^N (\text{p}(r(e_1, e_2) | \mathbf{w}^{(i)}))^{\alpha}, \quad [17.20]$$

9053 where $\text{p}(r(e_1, e_2) | \mathbf{w}^{(i)})$ is the probability of relation r between entities e_1 and e_2 conditioned
 9054 on the text $\mathbf{w}^{(i)}$, and $\alpha \gg 1$ is a tunable hyperparameter. Using this heuristic, it is
 9055 possible to rank all candidate relations, and trace out a **precision-recall curve** as more rel-
 9056ations are extracted.⁵ Alternatively, features can be aggregated across multiple passages
 9057 of text, feeding a single type-level relation extraction system (Wolfe et al., 2017).

9058 Precision can be improved by introducing constraints across multiple relations. For
 9059 example, if we are certain of the relation $\text{PARENT}(e_1, e_2)$, then it cannot also be the case
 9060 that $\text{PARENT}(e_2, e_1)$. Integer linear programming makes it possible to incorporate such
 9061 constraints into a global optimization (Li et al., 2011). Other pairs of relations have pos-
 9062itive correlations, such $\text{MAYOR}(e_1, e_2)$ and $\text{LIVED-IN}(e_1, e_2)$. Compatibility across relation
 9063 types can be incorporated into probabilistic graphical models (e.g., Riedel et al., 2010).

9064 17.2.3.2 Distant supervision

9065 Relation extraction is “annotation hungry,” because each relation requires its own la-
 9066 beled data. Rather than relying on annotations of individual documents, it would be
 9067 preferable to use existing knowledge resources — such as the many facts that are al-
 9068 ready captured in knowledge bases like DBpedia. However such annotations raise the
 9069 inverse of the information fusion problem considered above: the existence of the relation
 9070 $\text{MAYOR}(\text{Maynard Jackson Jr., Atlanta})$ provides only **distant supervision** for the
 9071 example texts in which this entity pair is mentioned.

9072 One approach is to treat the entity pair as the instance, rather than the text itself (Mintz
 9073 et al., 2009). Features are then aggregated across all sentences in which both entities are
 9074 mentioned, and labels correspond to the relation (if any) between the entities in a knowl-
 9075 edge base, such as FreeBase. Negative instances are constructed from entity pairs that are
 9076 not related in the knowledge base. In some cases, two entities are related, but the knowl-
 9077 edge base is missing the relation; however, because the number of possible entity pairs is
 9078 huge, these missing relations are presumed to be relatively rare. This approach is shown
 9079 in Figure 17.2.

9080 In **multiple instance learning**, sets of instances receive a single label, which applies
 9081 only to an unknown subset (Dietterich et al., 1997; Maron and Lozano-Pérez, 1998). This
 9082 formalizes the framework of distant supervision: the relation $\text{REL}(a, b)$ can act as a label

⁵The 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.

9083 for the entire set of sentences mentioning entities a and b , even when only a subset of
 9084 these sentences actually describes the relation. One approach to multi-instance learning
 9085 is to introduce a binary **latent variable** for each sentence, indicating whether the sen-
 9086 tence expresses the labeled relation (Riedel et al., 2010). A variety of inference techniques
 9087 have been employed for this probabilistic model of relation extraction: Surdeanu et al.
 9088 (2012) use expectation maximization, Riedel et al. (2010) use sampling, and Hoffmann
 9089 et al. (2011) use a custom graph-based algorithm. Expectation maximization and sampling
 9090 are surveyed in chapter 5, and are covered in more detail by Murphy (2012); graph-based
 9091 methods are surveyed by Mihalcea and Radev (2011).

9092 17.2.4 Open information extraction

9093 In classical relation extraction, the set of relations is defined in advance, using a **schema**.
 9094 The relation for any pair of entities can then be predicted using multi-class classification.
 9095 In **open information extraction**, a relation can be any triple of text. The example sentence

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.

9096 (17.13) instantiates several “relations” of this sort:

- 9097 • (*mayor of, Maynard Jackson, Atlanta*),
- 9098 • (*elected, Maynard Jackson, mayor of Atlanta*),
- 9099 • (*elected in, Maynard Jackson, 1973*),

9100 and so on. Extracting such tuples can be viewed as a lightweight version of **semantic role**
 9101 **labeling** (chapter 13), with only two argument types: first slot and second slot. The task is
 9102 generally evaluated on the relation level, rather than on the level of sentences: precision is
 9103 measured by the number of extracted relations that are accurate, and recall is measured by
 9104 the number of true relations that were successfully extracted. OpenIE systems are trained
 9105 from distant supervision or bootstrapping, rather than from labeled sentences.

9106 An early example is the TextRunner system (Banko et al., 2007), which identifies
 9107 relations with a set of handcrafted syntactic rules. The examples that are acquired from
 9108 the handcrafted rules are then used to train a speedier classification model that uses part-
 9109 of-speech patterns as features. Finally, the relations that are extracted by the classifier are
 9110 aggregated, removing redundant relations and computing the number of times that each
 9111 relation is mentioned in the corpus. TextRunner was the first in a series of systems that
 9112 performed increasingly accurate open relation extraction by incorporating more precise
 9113 linguistic features (Etzioni et al., 2011), distant supervision from Wikipedia infoboxes (Wu
 9114 and Weld, 2010), and better learning algorithms (Zhu et al., 2009).

9115 17.3 Events

9116 Relations link pairs of entities, but many real-world situations involve more than two en-
 9117 tities. Consider again the example sentence (17.13), which describes the **event** of an elec-
 9118 tion, with four properties: the office (`mayor`), the district (`Atlanta`), the date (`1973`), and

9119 the person elected (Maynard Jackson, Jr.). In **event detection**, a schema is provided
9120 for each event type (e.g., an election, a terrorist attack, or a chemical reaction), indicating
9121 all the possible properties of the event. The system is then required to fill in as many of
9122 these properties as possible (Doddington et al., 2004).

9123 Event detection systems generally involve a retrieval component (finding relevant
9124 documents and passages of text) and an extraction component (determining the proper-
9125 ties of the event based on the retrieved texts). Early approaches focused on finite-state pat-
9126 terns for identify event properties (Hobbs et al., 1997); such patterns can be automatically
9127 induced by searching for patterns that are especially likely to appear in documents that
9128 match the event query (Riloff, 1996). Contemporary approaches employ techniques that
9129 are similar to FrameNet semantic role labeling (§ 13.2), such as structured prediction over
9130 local and global features (Li et al., 2013) and bidirectional recurrent neural networks (Feng
9131 et al., 2016). These methods detect whether an event is described in a sentence, and if so,
9132 what are its properties.

9133 **Event coreference** Because multiple sentences may describe unique properties of a sin-
9134 gle event, **event coreference** is required to link event mentions across a single passage
9135 of text, or between passages (Humphreys et al., 1997). Bejan and Harabagiu (2014) de-
9136 fine event coreference as the task of identifying event mentions that share the same event
9137 participants (i.e., the slot-filling entities) and the same event properties (e.g., the time and
9138 location), within or across documents. Event coreference resolution can be performed us-
9139 ing supervised learning techniques in a similar way to entity coreference, as described
9140 in chapter 15: move left-to-right through the document, and use a classifier to decide
9141 whether to link each event reference to an existing cluster of coreferent events, or to cre-
9142 ate a new cluster (Ahn, 2006). Each clustering decision is based on the compatibility of
9143 features describing the participants and properties of the event. Due to the difficulty
9144 of annotating large amounts of data for entity coreference, unsupervised approaches are
9145 especially desirable (Chen and Ji, 2009; Bejan and Harabagiu, 2014). [todo: figure with
9146 example]

9147 **Relations between events** Just as entities are related to other entities, events may be
9148 related to other events: for example, the event of winning an election both *precedes* and
9149 *causes* the event of serving as mayor; moving to Atlanta *precedes* and *enables* the event of
9150 becoming mayor of Atlanta; moving from Dallas to Atlanta *prevents* the event of later be-
9151 coming mayor of Dallas. As these examples show, events may be related both temporally
9152 and causally. The **TimeML** annotation scheme specifies a set of six temporal relations
9153 between events (Pustejovsky et al., 2005), derived in part from **interval algebra** (Allen,
9154 1984). The TimeBank corpus provides TimeML annotations for 186 documents (Pust-
9155 ejovsky et al., 2003). Methods for detecting these temporal relations have combined super-
9156 vised machine learning with temporal constraints, such as transitivity (Mani et al., 2006;

	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.

9157 Chambers and Jurafsky, 2008).

9158 More recent annotation schemes and datasets have attempted to combine temporal
 9159 and causal relations (Mirza et al., 2014; Dunietz et al., 2017): for example, the CaTeRS
 9160 dataset includes annotations of 320 five-sentence short stories (Mostafazadeh et al., 2016).
 9161 Abstracting still further, **processes** are networks of causal relations between multiple
 9162 events. A small dataset of biological processes is annotated in the ProcessBank dataset (Be-
 9163 riant et al., 2014), with the goal of supporting automatic question answering on scientific
 9164 textbooks.

9165 17.4 Hedges, denials, and hypotheticals

9166 The methods described thus far apply to **propositions** about the way things are in the
 9167 real world. But natural language can also describe events and relations that are likely or
 9168 unlikely, possible or impossible, desired or feared. The following examples hint at the
 9169 scope of the problem (Prabhakaran et al., 2010):

- 9170 (17.17) GM will lay off workers.
- 9171 (17.18) A spokesman for GM said GM will lay off workers.
- 9172 (17.19) GM may lay off workers.
- 9173 (17.20) The politician claimed that GM will lay off workers.
- 9174 (17.21) Some wish GM would lay off workers.
- 9175 (17.22) Will GM lay off workers?
- 9176 (17.23) Many wonder whether GM will lay off workers.

9177 Accurate information extraction requires handling these **extra-propositional** aspects
 9178 of meaning, which are sometimes summarized under the terms **modality** and **negation**.⁶

⁶The 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

9179 Modality refers to expressions of the speaker's attitude towards her own statements, in-
9180 cluding "degree of certainty, reliability, subjectivity, sources of information, and perspec-
9181 tive" (Morante and Sporleder, 2012). Various systematizations of modality have been pro-
9182 posed (e.g., Palmer, 2001), including categories such as future, interrogative, imperative,
9183 conditional, and subjective. Information extraction is particularly concerned with nega-
9184 tion and certainty. For example, Saurí and Pustejovsky (2009) link negation with a modal
9185 calculus of certainty, likelihood, and possibility, creating the two-dimensional analysis
9186 shown in Table 17.4. This schema is the basis for the FactBank corpus, with annotations
9187 of the **factuality** of all sentences in 208 documents of news text.

9188 A related concept is **hedging**, in which speakers limit their commitment to a proposi-
9189 tion (Lakoff, 1973):

9190 (17.24) These results **suggest** that expression of c-jun, jun B and jun D genes **might** be in-
9191 volved in terminal granulocyte differentiation... (Morante and Daelemans, 2009)

9192 (17.25) A whale is **technically** a mammal (Lakoff, 1973)

9193 In the first example, the hedges *suggest* and *might* communicate uncertainty; in the second
9194 example, there is no uncertainty, but the hedge *technically* indicates that the evidence for
9195 the proposition will not fully meet the reader's expectations. Hedging has been studied
9196 extensively in scientific texts (Medlock and Briscoe, 2007; Morante and Daelemans, 2009),
9197 where the goal of large-scale extraction of scientific facts is obstructed by hedges and spec-
9198 ulation. Still another related aspect of modality is **evidentiality**, in which speakers mark
9199 the source of their information. In many languages, it is obligatory to mark evidentiality
9200 through affixes or particles (Aikhenvald, 2004); while evidentiality is not grammaticalized
9201 in English, authors are expected to express this information in contexts such as journal-
9202 ism (Kovach and Rosenstiel, 2014) and Wikipedia.⁷

9203 Methods for handling negation and modality generally include two phases:

- 9204 1. detecting negated or uncertain events;
9205 2. identifying the scope and focus of the negation or modal operator.

9206 A considerable body of work on negation has employed rule-based techniques such as
9207 regular expressions (Chapman et al., 2001) to detect negated events. Such techniques

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.

⁷<https://en.wikipedia.org/wiki/Wikipedia:Verifiability>

match lexical cues (e.g., *Norwood was **not** elected Mayor*), while avoiding “double negatives” (e.g., *surely all this is **not without** meaning*). More recent approaches employ classifiers over lexical and syntactic features (Uzuner et al., 2009) and sequence labeling (Prabhakaran et al., 2010).

The tasks of scope and focus resolution is more fine grained, as shown in the following example from Morante and Sporleder (2012):

- (17.26) [After his habit he said] **nothing**, and after mine I asked no questions.
 After his habit he said nothing, and [after mine I asked] **no** [questions].

In this sentence, there are two negation cues (*nothing* and *no*). Each negates an event, indicated by the underlined verbs *said* and *asked* (this is the focus of negation), and each occurs within a scope: *after his habit he said* and *after mine I asked* questions. These tasks are typically formalized as sequence labeling problems, with each word token labeled as beginning, inside, or outside of a cue, focus, or scope span (see § 8.3). Conventional sequence labeling approaches can then be applied, using surface features as well as syntax (Velldal et al., 2012) and semantic analysis (Packard et al., 2014). Labeled datasets include the BioScope corpus of biomedical texts (Vincze et al., 2008) and a shared task dataset of detective stories by Arthur Conan Doyle (Morante and Blanco, 2012).

17.5 Question answering and machine reading

The victory of the Watson question-answering system against three top human players on the game show *Jeopardy!* was a landmark moment in the history of natural language processing (Ferrucci et al., 2010). Game show questions are usually answered by **factoids**: entity names and short phrases.⁸ The task of factoid question answering is therefore closely related to information extraction, with the additional problem of accurately parsing the question.

17.5.1 Formal semantics

Semantic parsing is an effective method for question-answering in restricted domains such as questions about geography and airline reservations (Zettlemoyer and Collins, 2005), and has also been applied in “open-domain” settings such as question answering on Freebase (Berant et al., 2013) and biomedical research abstracts (Poon and Domingos, 2009). One approach is to convert the question into a lambda calculus expression that returns a boolean value: for example, the question *who is the mayor of the capital of Georgia?*

⁸The 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).

9239 would be converted to,

$$\lambda x. \exists y \text{ CAPITAL(Georgia, } y) \wedge \text{MAYOR}(y, x). \quad [17.21]$$

9240 This lambda expression can then be used to query an existing knowledge base, returning
 9241 all entities that satisfy it. The knowledge base itself could be constructed using techniques
 9242 described in § 17.2.3.

9243 17.5.2 Machine reading

9244 Recent work has focused on answering questions about specific textual passages, similar
 9245 to the reading comprehension examinations for young students (Hirschman et al., 1999).
 9246 This task has come to be known as **machine reading**.

9247 17.5.2.1 Datasets

9248 The machine reading problem can be formulated in a number of different ways. The most
 9249 important distinction is what form the answer should take.

- 9250 • **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,
 9251 the answer is deducible from the text alone, while in the science exams, the system
 9252 must make inferences using an existing model of the underlying scientific phenomena.
 9253 Here is an example from MCTest:

9255 (17.27) James the turtle was always getting into trouble. Sometimes he'd reach into
 9256 the freezer and empty out all the food ...

9257 Q: What is the name of the trouble making turtle?
 9258 (a) Fries
 9259 (b) Pudding
 9260 (c) James
 9261 (d) Jane

- 9262 • **Cloze-style “fill in the blank” questions**, as in the CNN/Daily Mail comprehension
 9263 task (Hermann et al., 2015), the Children’s Book Test (Hill et al., 2016), and the Who-
 9264 did-What dataset (Onishi et al., 2016). In these tasks, the system must guess which
 9265 word or entity completes a sentence, based on reading a passage of text. Here is an
 9266 example from Who-did-What:

9267 (17.28) Q: Tottenham manager Juande Ramos has hinted he will allow ____ to leave
 9268 if the Bulgaria striker makes it clear he is unhappy. (Onishi et al., 2016)

(c) Jacob Eisenstein 2018. Work in progress.

9269 The query sentence may be selected either from the story itself, or from an external
 9270 summary. In either case, datasets can be created automatically by processing large
 9271 quantities existing documents. An additional constraint is that that missing element
 9272 from the cloze must appear in the main passage of text: for example, in Who-did-
 9273 What, the candidates include all entities mentioned in the main passage. In the
 9274 CNN/Daily Mail dataset, each entity name is replaced by a unique identifier, e.g.,
 9275 entity37. This ensures that correct answers can only be obtained by accurately
 9276 reading the text, and not from external knowledge about the entities.

- 9277 • **Extractive** question answering, in which the answer is drawn from the original text.
 9278 In WikiQA, answers are sentences (Yang et al., 2015); in the Stanford Question An-
 9279 swering Dataset (SQuAD), answers are words or short phrases (Rajpurkar et al.,
 9280 2016):

9281 (17.29) In metereology, precipitation is any product of the condensation of atmo-
 9282 spheric water vapor that falls under gravity.
 9283 Q: What causes precipitation to fall? A: gravity

9284 In both WikiQA and SQuAD, the original texts are Wikipedia articles, and the ques-
 9285 tions are generated by crowdworkers.

9286 **17.5.2.2 Methods**

9287 A baseline method is to search the text for sentences or short passages that overlap with
 9288 both the query and the candidate answer (Richardson et al., 2013). In example (17.27),
 9289 this baseline would select the correct answer (c), since *James* appears in a sentence that
 9290 includes the query terms *trouble* and *turtle*.

This baseline can be implemented as a neural architecture, using an **attention mechanism** that 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).

$$\mathbf{h}^{(q)} = \text{BiLSTM}(\mathbf{w}^{(q)}; \Theta^{(q)}) \quad [17.22]$$

$$\mathbf{h}^{(p)} = \text{BiLSTM}(\mathbf{w}^{(p)}; \Theta^{(p)}). \quad [17.23]$$

The query is represented by vertically concatenating the final states of the left-to-right and right-to-left passes:

$$\mathbf{u} = [\overrightarrow{\mathbf{h}}^{(q)}_{M_q}; \overleftarrow{\mathbf{h}}^{(q)}_0]. \quad [17.24]$$

(c) Jacob Eisenstein 2018. Work in progress.

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,

$$\tilde{\alpha}_m = (\mathbf{u}^{(q)})^\top \mathbf{W}_a \mathbf{h}_m^{(p)} \quad [17.25]$$

$$\boldsymbol{\alpha} = \text{SoftMax}(\tilde{\boldsymbol{\alpha}}) \quad [17.26]$$

$$\mathbf{o} = \sum_{m=1}^M \alpha_m \mathbf{h}_m^{(p)}. \quad [17.27]$$

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,

$$\hat{c} = \underset{c}{\operatorname{argmax}} \mathbf{o} \cdot \mathbf{x}_c. \quad [17.28]$$

9291 This architecture can be trained end-to-end from a loss based on the log-likelihood of the
 9292 correct answer. A number of related architectures have been proposed (e.g., Hermann
 9293 et al., 2015; Kadlec et al., 2016; Dhingra et al., 2017; Cui et al., 2017), and the relationships
 9294 between these methods are surveyed by Wang et al. (2017).

9295 Additional reading

9296 The field of information extraction is surveyed in course notes by Grishman (2012), and
 9297 more recently in a short survey paper (Grishman, 2015). Shen et al. (2015) survey the task
 9298 of entity linking, and Ji and Grishman (2011) survey work on knowledge base popula-
 9299 tion. This chapter’s discussion of non-propositional meaning was strongly influenced by
 9300 Morante and Sporleder (2012), who introduced a special issue of the journal *Computational
 9301 Linguistics* dedicated to recent work on modality and negation.

9302 Exercises

9303 1. Consider the following heuristic for entity linking:

- 9304 • Among all entities that have the same type as the mention (e.g., LOC, PER),
 9305 choose the one whose name has the lowest edit distance from the mention.
- 9306 • If more than one entity has the right type and the lowest edit distance from the
 9307 mention, choose the most popular one.
- 9308 • If no candidate entity has the right type, choose NIL.

Now suppose you have the following feature function:

$$f(y, \mathbf{x}) = [\text{edit-dist}(\text{name}(y), \mathbf{x}), \text{same-type}(y, \mathbf{x}), \text{popularity}(y), \delta(y = \text{NIL})]$$

(c) Jacob Eisenstein 2018. Work in progress.

Design a set of ranking weights θ that match the heuristic. You may assume that edit distance and popularity are always in the range [0, 100], and that the NIL entity has values of zero for all features except δ ($y = \text{NIL}$).

2. Now consider another heuristic:

- Among all candidate entities that have edit distance zero from the mention and the right type, choose the most popular one.
- If no entity has edit distance zero from the mention, choose the one with the right type that is most popular, regardless of edit distance.
- If no entity has the right type, choose NIL.

Using the same features and assumptions from the previous problem, prove that there is no set of weights that could implement this heuristic. Then show that the heuristic can be implemented by adding a single feature. Your new feature should consider only the edit distance.

3. * Consider the following formulation for collective entity linking, which rewards sets of entities that are all of the same type, where “types” can be elements of any set:

$$\psi_c(\mathbf{y}) = \begin{cases} \alpha & \text{all entities in } \mathbf{y} \text{ have the same type} \\ \beta & \text{more than half of the entities in } \mathbf{y} \text{ have the same type} \\ 0 & \text{otherwise.} \end{cases} \quad [17.29]$$

Show how to implement this model of collective entity linking in an **integer linear program**. You may want to review § 13.2.2.

To get started, here is an integer linear program for entity linking, without including the collective term ψ_c :

$$\begin{aligned} \max_{z_{i,y} \in \{0,1\}} \quad & \sum_{i=1}^N \sum_{y \in \mathcal{Y}(\mathbf{x}^{(i)})} s_{i,y} z_{i,y} \\ \text{s.t.} \quad & \sum_{y \in \mathcal{Y}(\mathbf{x}^{(i)})} z_{i,y} \leq 1 \quad \forall i \in \{1, 2, \dots, N\} \end{aligned}$$

where $z_{i,y} = 1$ if entity y is linked to mention i , and $s_{i,y}$ is a parameter that scores the quality of this individual ranking decision, e.g., $s_{i,y} = \theta \cdot f(y, \mathbf{x}^{(i)}, \mathbf{c}^{(i)})$.

To incorporate the collective linking score, you may assume parameters r ,

$$r_{y,\tau} = \begin{cases} 1, & \text{entity } y \text{ has type } \tau \\ 0, & \text{otherwise.} \end{cases} \quad [17.30]$$

Hint: You will need to define several auxiliary variables to optimize over.

- 9331 4. Run `nltk.corpus.download('reuters')` to download the Reuters corpus in
 9332 NLTK, and run `from nltk.corpus import reuters` to import it. The com-
 9333 mand `reuters.words()` returns an iterator over the tokens in the corpus.
- 9334 a) Apply the pattern `_____`, such as `_____` to this corpus, obtaining candidates for
 9335 the IS-A relation, e.g. `IS-A(Romania, Country)`. What are three pairs that this
 9336 method identifies correctly? What are three different pairs that it gets wrong?
- 9337 b) Design a pattern for the PRESIDENT relation, e.g. `PRESIDENT(Philippines, Corazon Aquino)`.
 9338 In this case, you may want to augment your pattern matcher with the ability
 9339 to match multiple token wildcards, perhaps using case information to detect
 9340 proper names. Again, list three correct
- 9341 c) Preprocess the Reuters data by running a named entity recognizer, replacing
 9342 tokens with named entity spans when applicable. Apply your PRESIDENT
 9343 matcher to this new data. Does the accuracy improve? Compare 20 randomly-
 9344 selected pairs from this pattern and the one you designed in the previous part.
- 9345 5. Represent the dependency path $\mathbf{x}^{(i)}$ as a sequence of words and dependency arcs
 9346 of length M_i , ignoring the endpoints of the path. In example 1 of Table 17.2, the
 9347 dependency path is,

$$\mathbf{x}^{(1)} = (\xleftarrow[\text{NSUBJ}]{} \text{traveled}, \xrightarrow[\text{OBL}]{}) \quad [17.31]$$

9348 If $x_m^{(i)}$ is a word, then let $\text{pos}(x_m^{(i)})$ be its part-of-speech, using the tagset defined in
 9349 chapter 8.

We can define the following kernel function over pairs of dependency paths (Bunescu and Mooney, 2005):

$$\kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \begin{cases} 0, & M_i \neq M_j \\ \prod_{m=1}^{M_i} c(x_m^{(i)}, x_m^{(j)}), & M_i = M_j \end{cases}$$

$$c(x_m^{(i)}, x_m^{(j)}) = \begin{cases} 2, & x_m^{(i)} = x_m^{(j)} \\ 1, & x_m^{(i)} \text{ and } x_m^{(j)} \text{ are words and } \text{pos}(x_m^{(i)}) = \text{pos}(x_m^{(j)}) \\ 0, & \text{otherwise.} \end{cases}$$

9350 Using this kernel function, compute the kernel similarities of example 1 from Ta-
 9351 ble 17.2 with the other five examples.

- 9352 6. Continuing from the previous problem, suppose that the instances have the follow-
 9353 ing labels:

$$y_2 = 1, y_3 = -1, y_4 = -1, y_5 = 1, y_6 = 1 \quad [17.32]$$

Identify the conditions for α and b under which $\hat{y}_1 = 1$. Remember the constraint
 that $\alpha_i \geq 0$ for all i .

9354

Chapter 18

9355

Machine translation

9356 Machine translation (MT) is one of the “holy grail” problems in artificial intelligence, and
9357 it has the potential to transform society by facilitating communication between people
9358 anywhere in the world. As a result, MT has received a lot of attention and funding since
9359 the early 1950s. However, it has proved incredibly challenging, and while there has been
9360 substantial progress towards usable MT systems — especially for high-resource language
9361 pairs like English-French — we are still far from automatically producing translations that
9362 capture the nuance and depth of human language.

9363

18.1 Machine translation as a task

9364 Like most problems in natural language processing, machine translation can be viewed
9365 as optimization:

$$\hat{\mathbf{w}}^{(t)} = \operatorname{argmax}_{\mathbf{w}^{(t)}} \Psi(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}), \quad [18.1]$$

9366 where $\mathbf{w}^{(s)}$ is a sentence in a **source** language, $\mathbf{w}^{(t)}$ is a sentence in the **target language**,
9367 and Ψ is a scoring function. As usual, this formalism requires two components: a decod-
9368 ing algorithm for computing $\hat{\mathbf{w}}^{(t)}$, and a learning algorithm for estimating the parameters
9369 of the scoring function Ψ .

9370 Decoding is also difficult for machine translation, because of the huge space of possible
9371 translations. We have faced large label spaces before: for example, in sequence labeling,
9372 the set of possible label sequences is exponential in the length of the input. In these cases,
9373 it was possible to search the space quickly by introducing locality assumptions: for exam-
9374 ple, that a single tag depends only on its predecessor, or that a single production depends
9375 only on its parent. In machine translation, no such locality assumptions seem to be possi-
9376 ble. Human translators reword, reorder, and rearrange words; they replace single words
9377 with multi-word phrases, and vice versa. This flexibility means that in even relatively

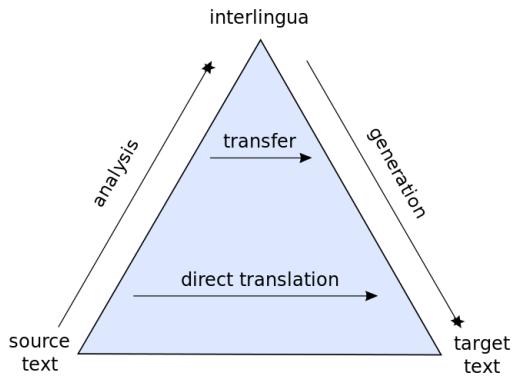


Figure 18.1: The Vauquois Pyramid

9378 simple translation models, decoding is NP-hard (Knight, 1999). Approaches for dealing
 9379 with this complexity are described in § 18.4.

Estimation is complicated because labeled translation data usually comes in the form of **bitext**, or aligned sentences, e.g.,

$$\begin{aligned} w^{(s)} &= A \text{ Vinay le gusta las manzanas.} \\ w^{(t)} &= \text{Vinay likes apples.} \end{aligned}$$

9380 A useful feature function would note the translation pairs (*gusta, likes*), (*manzanas, apples*),
 9381 and even (*Vinay, Vinay*). But this word-to-word **alignment** is not given in the data. One
 9382 solution is to treat this alignment as a **latent variable**; this is the approach taken by clas-
 9383 sical **statistical machine translation** (SMT) systems, described in § 18.2. Another solution
 9384 is to model the relationship between $w^{(t)}$ and $w^{(s)}$ through a more complex and expres-
 9385 sive function; this is the approach taken by **neural machine translation** (NMT) systems,
 9386 described in § 18.3.

9387 The **Vauquois Pyramid** is a theory of how translation should be modeled. At the low-
 9388 est level, we translate individual words, but the horizontal distance at this level is large,
 9389 because languages express ideas differently. If we can move up the triangle to syntac-
 9390 tic structure, the distance for translation is reduced; we then need only produce target-
 9391 language text from the syntactic representation, which can be as simple as reading off a
 9392 tree. Further up the triangle lies semantics; translating between semantic representations
 9393 should be easier still, but mapping between semantics and surface text is a difficult, un-
 9394 solved problem. At the top of the triangle is **interlingua**, a semantic representation that is
 9395 so generic, it is identical across all human languages. Philosophers debate whether such
 9396 a thing as interlingua is really possible (Derrida, 1985). While the first-order logic repre-
 9397 sentations discussed in chapter 12 might be considered to be language independent, 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*.

set of logical relations is suspiciously similar to a subset of English words (Nirenburg and Wilks, 2001). Nonetheless, the idea of linking translation and semantic understanding may still be a promising path, if the resulting translations better preserve the meaning of the original text.

18.1.1 Evaluating translations

There are two main criteria for a translation.

- **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.

These criteria are summarized in Table 18.1.

Automated evaluations of machine translations typically merge both of these criteria, by comparing the hypothesized 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 the n -gram precision: what fraction of n -grams in the system translation appear in the reference? Specifically, for each n -gram length, the precision is defined as,

$$p_n = \frac{\text{number of } n\text{-grams appearing in both reference and hypothesis translations}}{\text{number of } n\text{-grams appearing in the hypothesis translation}}. \quad [18.2]$$

The n -gram precisions for three hypothesis translations are shown in Figure 18.2.

¹A gloss is a word-for-word translation.

	Translation	p_1	p_2	p_3	p_4	BP	BLEU
<i>Reference</i>	<i>Vinay likes programming in Python</i>						
<i>Sys1</i>	<i>To Vinay it like to program Python</i>	$\frac{2}{7}$	0	0	0	1	.21
<i>Sys2</i>	<i>Vinay likes Python</i>	$\frac{3}{3}$	$\frac{1}{2}$	0	0	.51	.33
<i>Sys3</i>	<i>Vinay likes programming in his pajamas</i>	$\frac{4}{6}$	$\frac{3}{5}$	$\frac{2}{4}$	$\frac{1}{3}$	1	.76

Figure 18.2: A reference translation and three system outputs. For each output, p_n indicates the precision at each n -gram, and BP indicates the brevity penalty.

9419 The BLEU score is then based on the average, $\exp \frac{1}{N} \sum_{n=1}^N \log p_n$. Two modifications of
 9420 Equation 18.2 are necessary: (1) to avoid computing $\log 0$, all precisions are smoothed to
 9421 ensure that they are positive; (2) each n -gram in the source can be used at most once, so
 9422 that *to to to to to* does not achieve $p_1 = 1$ as a translation for *to be or not to be*. Further-
 9423 more, precision-based metrics are biased in favor of short translations, which can achieve
 9424 high scores by simply minimizing the denominator in [18.2]. To avoid this issue, a **brevity**
 9425 **penalty** is applied to translations that are shorter than the reference. This penalty is indi-
 9426 cated as “BP” in Figure 18.2.

9427 Automated metrics like BLEU have been validated by correlation with human judg-
 9428 ments of translation quality. Nonetheless, it is not difficult to construct examples in which
 9429 the BLEU score is high, yet the translation is disfluent or carries a completely different
 9430 meaning from the original. To give just one example, consider the problem of translating
 9431 pronouns. Because pronouns refer to specific entities, a single incorrect pronoun can obliti-
 9432 onate the semantics of the original sentence. Existing state-of-the-art systems generally
 9433 do not attempt the reasoning necessary to correctly resolve pronominal anaphora (Hard-
 9434 meier, 2012). Despite the importance of pronouns for preserving meaning, they have a
 9435 marginal impact on BLEU, which may help to explain why existing systems do not make
 9436 a greater effort to translate them correctly!

9437 **Fairness and bias** The problem of pronoun translation intersects with issues of fairness
 9438 and bias. In many languages, such as Turkish, the third person singular pronoun is gender
 9439 neutral. Today’s state-of-the-art systems produce the following Turkish-English transla-
 9440 tions (Caliskan et al., 2017):

- 9441 (18.1) *O bir doktor.*
 He is a doctor.
 9442 (18.2) *O bir hemşire.*
 She is a nurse.

9443 The same problem arises for other professions that have stereotypical genders, such as
 9444 engineers, soldiers, and teachers, and for other languages that have gender-neutral pro-
 9445 nouns. This bias was not directly programmed into the translation model; it arises from
 9446 statistical tendencies in existing datasets. This highlights a general problem with data-
 9447 driven approaches, which can perpetuate biases that negatively impact disadvantaged
 9448 groups. Worse, machine learning can *amplify* biases in data (Bolukbasi et al., 2016): if a
 9449 dataset has even a slight tendency towards men as doctors, the resulting translation model
 9450 may produce translations in which doctors are always *he*, and nurses are always *she*.

9451 **Other metrics** A range of other automated metrics have been proposed for machine
 9452 translation. One potential weakness of BLEU is that it only measures precision; METEOR
 9453 is a weighted *F*-MEASURE, which is a combination of recall and precision (see § 4.4.1).
 9454 **Translation Error Rate (TER)** computes the string edit distance between the reference and
 9455 the hypothesis (Snover et al., 2006). For language pairs like English and Japanese, there
 9456 are substantial differences in word order, and word order errors are not sufficiently cap-
 9457 tured by *n*-gram based metrics. The RIBES metric applies rank correlation to measure
 9458 the similarity in word order between the system and reference translations (Isozaki et al.,
 9459 2010).

9460 18.1.2 Data

9461 Data-driven approaches to machine translation rely primarily on **parallel corpora**: sentence-
 9462 level translations. Early work focused on government records, in which fine-grained
 9463 official translations are often required. For example, the IBM translation systems were
 9464 based on the proceedings of the Canadian Parliament, called **Hansards**, which recorded
 9465 in English and French Brown et al. (1990). The growth of the European Union led to the
 9466 development of the **EuroParl corpus**, which spans 21 European languages. While these
 9467 datasets helped to launch the field of machine translation, they are restricted to narrow
 9468 domains and a formal speaking style, limiting their applicability to other types of text. As
 9469 more resources are committed to machine translation, new translation datasets have been
 9470 commissioned. This has broadened the scope of available data to news,² movie subtitles,³
 9471 social media (Ling et al., 2013), dialogues (Fordyce, 2007), TED talks (Paul et al., 2010),
 9472 and scientific research articles (Nakazawa et al., 2016).

9473 For most languages, the only source of translation data is the Christian Bible (Resnik
 9474 et al., 1999). While relatively small, at less than a million tokens, the Bible has been
 9475 translated into more than 2000 languages, far outpacing any other corpus. Many lan-
 9476 guages have sizable parallel corpora with some high-resource language, but not with each

²https://catalog.ldc.upenn.edu/LDC2010T10_translation-task.html ³<http://www.statmt.org/wmt15/>

³<http://opus.nlpl.eu/>

other. The high-resource language can then be used as a “pivot” or “bridge” (Boitet, 1988; Utiyama and Isahara, 2007): for example, De Gispert and Marino (2006) use Spanish as a bridge for translation between Catalan and English.

The main bottleneck in machine translation data is the need for parallel corpora that are aligned at the sentence level. Some work has explored the possibility of creating such corpora automatically from parallel texts, such as web pages and Wikipedia articles (Kilgarriff and Grefenstette, 2003; Resnik and Smith, 2003; Adafre and De Rijke, 2006). Another approach is to attempt to create large parallel corpora through crowdsourcing (Zaidan and Callison-Burch, 2011).

18.2 Statistical machine translation

The previous section introduced adequacy and fluency as the two main criteria for machine translation. A natural approach would be to represent them with separate scores,

$$\Psi(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \Psi_A(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) + \Psi_F(\mathbf{w}^{(t)}). \quad [18.3]$$

The fluency score Ψ_F need not even consider the source sentence; it only judges $\mathbf{w}^{(t)}$ on whether it is fluent in the target language.

This decomposition is advantageous because it makes it possible to estimate the two scoring functions on separate data. While the adequacy model must be estimated from bitext — which is relatively expensive and rare — the fluency model can be estimated from monolingual text in the target language. Large monolingual corpora are now available in many languages, thanks to resources such as Wikipedia.

An elegant justification of the decomposition in Equation 18.3 is provided by the **noisy channel model**, in which each scoring function is a log probability:

$$\Psi_A(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) \triangleq \log p_{S|T}(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}) \quad [18.4]$$

$$\Psi_F(\mathbf{w}^{(t)}) \triangleq \log p_T(\mathbf{w}^{(t)}) \quad [18.5]$$

$$\Psi(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \log p_{S|T}(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}) + \log p_T(\mathbf{w}^{(t)}) = \log p_{S,T}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}). \quad [18.6]$$

By setting the scoring functions equal to the logarithms of the prior and likelihood, their sum is equal to $\log p_{S,T}$, which is the logarithm of the joint probability of the source and target. The sentence $\hat{\mathbf{w}}^{(t)}$ that maximizes this joint probability is also the maximizer of the conditional probability $p_{T|S}$, making it the most likely target language sentence, conditioned on the source:

$$p_{T|S}(\mathbf{w}^{(t)} | \mathbf{w}^{(s)}) = \frac{p_{S,T}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)})}{p_S(\mathbf{w}^{(s)})} \quad [18.7]$$

$$\operatorname{argmax}_{\mathbf{w}^{(t)}} \Psi(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \operatorname{argmax}_{\mathbf{w}^{(t)}} \log p_{S,T}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \operatorname{argmax}_{\mathbf{w}^{(t)}} \log p_{T|S}(\mathbf{w}^{(t)} | \mathbf{w}^{(s)}). \quad [18.8]$$

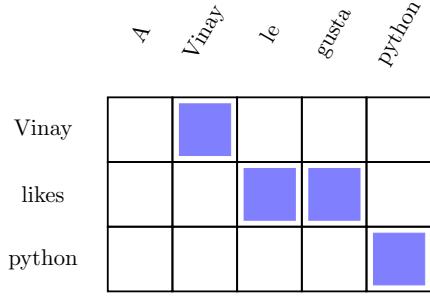


Figure 18.3: An example word-to-word alignment

9496 The noisy channel model can be justified by a generative story. The target text is origi-
 9497 nally generated from a probability model p_T . It is then encoded in a “noisy channel” $p_{S|T}$,
 9498 which converts it to a string in the source language. In decoding, we apply Bayes’ rule
 9499 to recover the string $w^{(t)}$ that is maximally likely under the conditional probability $p_{T|S}$.
 9500 Under this interpretation, the target probability p_T is just a language model, and can be
 9501 estimated using any of the techniques from chapter 6. The only remaining problem is to
 9502 estimate the translation model $p_{S|T}$.

9503 18.2.1 Statistical translation modeling

9504 The simplest decomposition of the translation model is word-to-word: each word in the
 9505 source string should be aligned to a word in the translation. In this approach, we need
 9506 an **alignment** $\mathcal{A}(w^{(s)}, w^{(t)})$, which contains a list of pairs of source and target tokens. For
 9507 example, given $w^{(s)} = A\ Vinay\ le\ gusta\ Python$ and $w^{(t)} = Vinay\ likes\ Python$, one possible
 9508 word-to-word alignment is,

$$\mathcal{A}(w^{(s)}, w^{(t)}) = \{(A, \emptyset), (Vinay, Vinay), (le, likes), (gusta, likes), (Python, Python)\}. \quad [18.9]$$

9509 This alignment is shown in Figure 18.3. Another, less promising, alignment is:

$$\mathcal{A}(w^{(s)}, w^{(t)}) = \{(A, Vinay), (Vinay, likes), (le, Python), (gusta, \emptyset), (Python, \emptyset)\}. \quad [18.10]$$

9510 Each alignment contains exactly one tuple for each word in the *source*, which serves to
 9511 explain how the source word could be translated from the target, as required by the trans-
 9512 lation probability $p_{S|T}$. If no appropriate word in the target can be identified for a source
 9513 word, it is aligned to \emptyset — as is the case for the function word *a* in the example, which
 9514 glosses to the English word *to*. Words in the target can align with multiple words in the
 9515 source, so that the target word *likes* can align to both *le* and *gusta* in the source.

Given the alignment, we can define the translation probability as,

$$p(\mathbf{w}^{(s)}, \mathcal{A} | \mathbf{w}^{(t)}) = \prod_{m=1}^{M^{(s)}} p(w_m^{(s)}, a_m | w_{a_m}^{(t)}, m, M^{(s)}, M^{(t)}) \quad [18.11]$$

$$= \prod_{m=1}^{M^{(s)}} p(a_m | m, M^{(s)}, M^{(t)}) \times p(w_m^{(s)} | w_{a_m}^{(t)}). \quad [18.12]$$

9516 This probability model makes two key assumptions:

- 9517 • The alignment probability factors across tokens,

$$p(\mathcal{A} | \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \prod_{m=1}^{M^{(s)}} p(a_m | m, M^{(s)}, M^{(t)}). \quad [18.13]$$

9518 This means that each alignment decision is independent of the others, and depends
9519 only on the index m , and the sentence lengths $M^{(s)}$ and $M^{(t)}$.

- 9520 • The translation probability also factors across tokens,

$$p(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}, \mathcal{A}) = \prod_{m=1}^{M^{(s)}} p(w_m^{(s)} | w_{a_m}^{(t)}), \quad [18.14]$$

9521 so that each word in $\mathbf{w}^{(s)}$ depends only on its aligned word in $\mathbf{w}^{(t)}$. This means that
9522 translation is word-to-word, ignoring context. The hope is that the target language
9523 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,

$$p(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \sum_{\mathcal{A}} p(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}, \mathcal{A}) \quad [18.15]$$

$$= p(\mathbf{w}^{(t)}) \sum_{\mathcal{A}} p(\mathcal{A}) p(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}, \mathcal{A}) \quad [18.16]$$

$$\geq p(\mathbf{w}^{(t)}) \max_{\mathcal{A}} p(\mathcal{A}) p(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}, \mathcal{A}). \quad [18.17]$$

The term $p(\mathcal{A})$ defines the prior probability over alignments. A series of alignment models with increasingly relaxed independence assumptions was produced by researchers at IBM in the 1980s and 1990s, known as IBM Models 1-6 (Och and Ney, 2003). IBM Model 1 makes the strongest independence assumption:

$$p(a_m | m, M^{(s)}, M^{(t)}) = \frac{1}{M^{(t)}}. \quad [18.18]$$

9524 In this model, every alignment is equally likely. This is almost surely wrong, but it re-
 9525 sults in a convex learning objective, yielding a good initialization for the more complex
 9526 alignment models (Brown et al., 1993; Koehn, 2009).

9527 18.2.2 Estimation

9528 Let us define the parameter $\theta_{u \rightarrow v}$ as the probability of translating target word u to source
 9529 word v . If word-to-word alignments were annotated, these probabilities could be com-
 9530 puted from relative frequencies,

$$\hat{\theta}_{u \rightarrow v} = \frac{\text{count}(u, v)}{\text{count}(u)}, \quad [18.19]$$

9531 where $\text{count}(u, v)$ is the count of instances in which word v was aligned to word u in
 9532 the training set, and $\text{count}(u)$ is the total count of the target word u . The smoothing
 9533 techniques mentioned in chapter 6 can help to reduce the variance of these probability
 9534 estimates.

9535 Conversely, if we had an accurate translation model, we could estimate the likelihood
 9536 of each alignment decision,

$$q_m(a_m \mid \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) \propto p(a_m \mid m, M^{(s)}, M^{(t)}) \times p(w_m^{(s)} \mid w_{a_m}^{(t)}), \quad [18.20]$$

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*,

$$\hat{\theta}_{u \rightarrow v} = \frac{E_q [\text{count}(u, v)]}{\text{count}(u)} \quad [18.21]$$

$$E_q [\text{count}(u, v)] = \sum_m q_m(a_m \mid \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) \delta(w_m^{(s)} = v) \delta(w_{a_m}^{(t)} = u). \quad [18.22]$$

9537 The **expectation-maximization** (EM) algorithm proceeds by iteratively updating q_m
 9538 and $\hat{\Theta}$. The algorithm is described in general form in chapter 5. For statistical machine
 9539 translation, the steps of the algorithm are:

9540 **E-step** Update beliefs about word alignment using Equation 18.20.

9541 **M-step** Update the translation model using Equations 18.21 and 18.22.

9542 In general, the expectation maximization algorithm is guaranteed to converge, but not to
 9543 a global optimum.

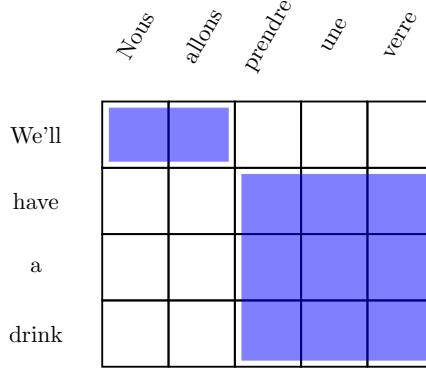


Figure 18.4: A phrase-based alignment between French and English, corresponding to example (18.3)

9544 18.2.3 Phrase-based translation

9545 Real translations are not word-to-word substitutions. One reason is that many multiword
9546 expressions are not translated literally, as shown in this example from French:

- 9547 (18.3) *Nous allons prendre un verre*
 We will take a glass
 9548 We'll have a drink

9549 The line *we will take a glass* is the word-for-word **gloss** of the French sentence; the transla-
 9550 tion *we'll have a drink* is shown on the third line. Such examples are difficult for word-to-
 9551 word translation models, since they require translating *prendre* to *have* and *verre* to *drink*.
 9552 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. (Note that these “phrases” are not necessarily syntactic constituents like the noun phrases and verb phrases described in chapter 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

$$p(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}, \mathcal{A}) = \prod_{((i, j), (k, \ell)) \in \mathcal{A}} p_{w^{(s)}|w^{(t)}}(\{w_{i+1}^{(s)}, w_{i+2}^{(s)}, \dots, w_j^{(s)}\} | \{w_{k+1}^{(t)}, w_{k+2}^{(t)}, \dots, w_\ell^{(t)}\}). \quad [18.23]$$

9553 The phrase alignment $((i, j), (k, \ell))$ indicates that the span $w_{i+1:j}^{(s)}$ is the translation of the
 9554 span $w_{k+1:\ell}^{(t)}$. An example phrasal alignment is shown in Figure 18.4. Note that the align-
 9555 ment set \mathcal{A} is required to cover all of the tokens in the source, just as in word-based trans-
 9556 lation. The probability model $p_{w^{(s)}|w^{(t)}}$ must now include translations for all phrase pairs,
 9557 which can be learned from expectation-maximization just as in word-based statistical ma-
 9558 chine translation.

9559 18.2.4 *Syntax-based translation

9560 The Vauquois Pyramid (Figure 18.1) suggests that translation might be easier if we are
 9561 able to take a “higher-level” view. One such possibility is to incorporate the syntactic
 9562 structure of the source, the target, or both. This is particularly promising for language
 9563 pairs that have consistent syntactic structures. For example, English adjectives almost al-
 9564 ways precede the nouns that they modify, while in Romance languages such as French and
 9565 Spanish, the adjective often follows the noun: thus, *angry fish* would translate to *pez (fish)*
 9566 *enojado (angry)* in Spanish. In word-to-word translation, these reorderings cause the align-
 9567 ment model to be overly permissive. It is not that the order of *any* pair of English words
 9568 can be reversed when translating into Spanish, but only adjectives and nouns within a
 9569 noun phrase. Similar issues arise when translating between verb-final languages such as
 9570 Japanese (in which verbs usually follow the subject and object), verb-initial languages like
 9571 Tagalog and classical Arabic, and verb-medial languages such as English.

9572 One elegant solution is to link parsing and translation in a **synchronous context-free**
 9573 **grammar** (SCFG; Chiang, 2007).⁴ An SCFG is a set of productions of the form $X \rightarrow (\alpha, \beta, \sim)$,
 9574 where X is a non-terminal, α and β are sequences of terminals or non-terminals, and \sim
 9575 is a one-to-one alignment of items in α with items in β . To handle the English-Spanish
 9576 adjective-noun ordering, an SCFG would include productions such as,

$$\text{NP} \rightarrow (\text{DET}_1 \text{NN}_2 \text{JJ}_3, \quad \text{DET}_1 \text{JJ}_3 \text{NN}_2), \quad [18.24]$$

9577 with subscripts indicating the alignment between the Spanish (left) and English (right)
 9578 parts of the right-hand side. Terminal productions yield translation pairs,

$$\text{JJ} \rightarrow (enojado_1, angry_1). \quad [18.25]$$

9579 A synchronous derivation begins with the start symbol S , and derives a pair of sequences
 9580 of terminal symbols.

9581 Given an SCFG in which each production yields at most two symbols (Chomsky Nor-
 9582 mal Form; see § 9.2.1.2), a target sentence can be parsed using only the CKY algorithm
 9583 (chapter 10). The resulting derivation also includes productions on the target side, and

⁴Key earlier work includes syntax-driven transduction (Lewis II and Stearns, 1968) and stochastic inver-
 sion transduction grammars (Wu, 1997).

9584 therefore yields a translation into the source sentence. Therefore, SCFGs make translation
 9585 very similar to parsing. In a weighted SCFG, the log probability $\log p_{S|T}$ can be computed
 9586 from the sum of the log-probabilities of the productions. However, combining SCFGs with
 9587 target language model is computationally expensive, necessitating approximate search al-
 9588 gorithms.

9589 Synchronous context-free grammars are an example of **tree-to-tree translation**, be-
 9590 cause they model the syntactic structure of both the target and source language. In **string-**
 9591 **to-tree translation**, string elements are translated into constituent tree fragments, which
 9592 are then assembled into a translation (Yamada and Knight, 2001; Galley et al., 2004); in
 9593 **tree-to-string translation**, the source side is parsed, and then transformed into a string on
 9594 the target side (Liu et al., 2006). A key question for syntax-based translation is the extent
 9595 to which we phrasal constituents align across translations (Fox, 2002), because this gov-
 9596 erns the extent to which we can rely on monolingual parsers and treebanks. For more on
 9597 syntax-based machine translation, see the monography by Williams et al. (2016).

9598 18.3 Neural machine translation

Neural network models to machine translation are based on the **encoder-decoder** architecture (Cho et al., 2014). The encoder network converts the source language sentence into a vector or matrix representation; the decoder network then converts the encoding into a sentence in the target language.

$$z = \text{ENCODE}(\mathbf{w}^{(s)}) \quad [18.26]$$

$$\mathbf{w}^{(t)} \mid \mathbf{w}^{(s)} \sim \text{DECODE}(z), \quad [18.27]$$

9599 where the second line means that the function $\text{DECODE}(z)$ defines the conditional proba-
 9600 bility $p(\mathbf{w}^{(t)} \mid \mathbf{w}^{(s)})$.

The decoder is typically a recurrent neural network, which generates the target lan-
 guage sentence one word at a time, while recurrently updating a hidden state. The en-
 coder and decoder networks are trained end-to-end from bitext. If the output layer of the
 decoder is a logistic function, then the entire architecture can be trained with the objective
 of maximizing the conditional log-likelihood,

$$\log p(\mathbf{w}^{(t)} \mid \mathbf{w}^{(s)}) = \sum_{m=1}^{M^{(t)}} p(w_m^{(t)} \mid \mathbf{w}_{1:m-1}^{(t)}, z) \quad [18.28]$$

$$p(w_m^{(t)} \mid \mathbf{w}_{1:m-1}^{(t)}, \mathbf{w}^{(s)}) \propto \exp \left(\boldsymbol{\beta}_{w_m^{(t)}} \cdot \mathbf{h}_{m-1}^{(t)} \right) \quad [18.29]$$

where the hidden state $\mathbf{h}_{m-1}^{(t)}$ is a recurrent function of the previously generated text

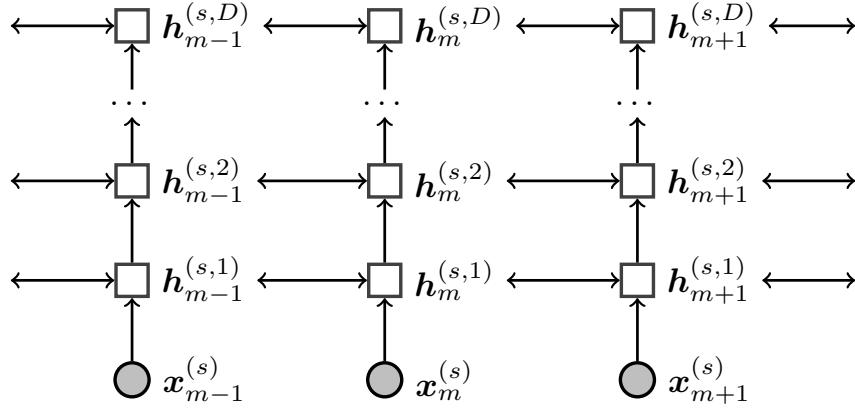


Figure 18.5: A deep bidirectional LSTM encoder

$w_{1:m-1}^{(t)}$ and the encoding z . The second line is equivalent to writing,

$$w_m^{(t)} \mid w_{1:m-1}^{(t)}, \mathbf{w}^{(s)} \sim \text{SoftMax}(\boldsymbol{\beta} \cdot \mathbf{h}_{m-1}^{(t)}), \quad [18.30]$$

9601 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
 9602 language vocabulary.

The simplest encoder-decoder architecture is the **sequence-to-sequence** model (Sutskever et al., 2014). In this model, the encoder is simply the final hidden state of a **long short-term memory (LSTM)** (see § 6.3.3) on the source sentence:

$$\mathbf{h}_m^{(s)} = \text{LSTM}(\mathbf{x}_m^{(s)}, \mathbf{h}_{m-1}^{(s)}) \quad [18.31]$$

$$z \triangleq \mathbf{h}_{M^{(s)}}^{(s)}, \quad [18.32]$$

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:

$$\mathbf{h}_0^{(t)} = z \quad [18.33]$$

$$\mathbf{h}_m^{(t)} = \text{LSTM}(\mathbf{x}_m^{(t)}, \mathbf{h}_{m-1}^{(t)}), \quad [18.34]$$

9603 where $\mathbf{x}_m^{(t)}$ is the embedding of the target language word $w_m^{(t)}$.

9604 Sequence-to-sequence translation is nothing more than wiring together two LSTMs:
 9605 one to read the source, and another to generate the target. To make the model work well,
 9606 some additional tweaks are needed:

- 9607 • Most notably, the model works much better if the source sentence is reversed, reading
 9608 from the end of the sentence back to the beginning. In this way, the words at the
 9609 beginning of the source have the greatest impact on the encoding z , and therefore
 9610 impact the words at the beginning of the target sentence. Later work on more ad-
 9611 vanced encoding models, such as **neural attention** (see § 18.3.1), has eliminated the
 9612 need for reversing the source sentence.
- 9613 • The encoder and decoder can be implemented as **deep LSTMs**, with multiple layers
 9614 of hidden states. As shown in Figure 18.5, each hidden state $\mathbf{h}_m^{(s,i)}$ at layer i is treated
 9615 as the input to an LSTM at layer $i + 1$:

$$\mathbf{h}_m^{(s,1)} = \text{LSTM}(\mathbf{x}_m^{(s)}, \mathbf{h}_{m-1}^{(s)}) \quad [18.35]$$

$$\mathbf{h}_m^{(s,i+1)} = \text{LSTM}(\mathbf{h}_m^{(s,i)}, \mathbf{h}_{m-1}^{(s)}), \quad \forall i \geq 1. \quad [18.36]$$

- 9613 The original work on sequence-to-sequence translation used four layers; in 2016,
 9614 Google’s commercial machine translation system used eight layers (Wu et al., 2016).⁵
- 9615 • Significant improvements can be obtained by creating an **ensemble** of translation
 9616 models, each trained from a different random initialization. For an ensemble of size
 9617 N , the per-token decoding probability is set equal to,

$$p(w^{(t)} | z, \mathbf{w}_{1:m-1}^{(t)}) = \frac{1}{N} \sum_{i=1}^N p_i(w^{(t)} | z, \mathbf{w}_{1:m-1}^{(t)}), \quad [18.37]$$

- 9618 where p_i is the decoding probability for model i . Each translation model in the
 9619 ensemble includes its own encoder and decoder networks.
- 9620 • The original sequence-to-sequence model used a fairly standard training setup: stochas-
 9621 tic gradient descent with an exponentially decreasing learning rate after the first five
 9622 epochs; mini-batches of 128 sentences, chosen to have similar length so that each
 9623 sentence on the batch will take roughly the same amount of time to process; gradi-
 9624 ent clipping (see § 3.3.4) to ensure that the norm of the gradient never exceeds some
 9625 predefined value.

9626 18.3.1 Neural attention

9627 The sequence-to-sequence model discussed in the previous section is a radical departure
 9628 from statistical machine translation, in which each word or phrase in the target language
 9629 is conditioned on a single word or phrase in the source language. Both approaches have
 9630 advantages. Statistical translation leverages the idea of compositionality — translations

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.

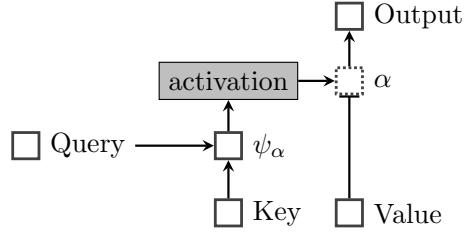


Figure 18.6: A general view of neural attention. The dotted box indicates that each $\alpha_{m \rightarrow n}$ can be viewed as a **gate** on value n .

of large units should be based on the translations of their component parts — and this seems crucial if we are to scale translation to longer units of text. But the translation of each word or phrase often depends on the larger context, and encoder-decoder models capture this context at the sentence level.

Is it possible for translation to be both contextualized and compositional? One approach is to augment neural translation with an **attention mechanism**. The idea of neural attention was described in § 17.5, but its application to translation bears further discussion. In general, attention can be thought of as using a query to select from a memory of key-value pairs. However, the query, keys, and values are all vectors, and the entire operation is differentiable. For each key n in the memory, we compute a score $\psi_\alpha(m, n)$ with respect to the query m . That score is a function of the compatibility of the key and the query, and can be computed using a small feedforward neural network. The vector of scores is passed through an activation function, such as softmax. The output of this activation function is a vector of non-negative numbers $[\alpha_{m \rightarrow 1}, \alpha_{m \rightarrow 2}, \dots, \alpha_{m \rightarrow N}]^\top$, with length N equal to the size of the memory. Each value in the memory v_n is multiplied by the attention $\alpha_{m \rightarrow n}$; the sum of these scaled values is the output. This process is shown in Figure 18.6. In the extreme case that $\alpha_{m \rightarrow n} = 1$ and $\alpha_{m \rightarrow n'} = 0$ for all other n' , then the attention mechanism simply selects the value v_n from the memory.

We now apply neural attention to the encoder-decoder architecture for translation. Rather than encoding the entire source sentence into a fixed length vector z , it can be encoded into a variable width matrix $Z \in \mathbb{R}^{K \times M^{(S)}}$, where K is the dimension of the hidden state, and $M^{(S)}$ is the length of 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, we access the attentional state by executing a query, which

is equal to the state of the decoder, $\mathbf{h}_m^{(t)}$. The compatibility scores can be computed as,

$$\psi_\alpha(m, n) = \mathbf{v}_\alpha \cdot \tanh(\Theta_\alpha[\mathbf{h}_m^{(t)}; \mathbf{h}_n^{(s)}]). \quad [18.38]$$

The function ψ is thus a two layer feedforward neural network, with weights \mathbf{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 activation:

Softmax attention

$$\alpha_{m \rightarrow n} = \frac{\exp \psi_\alpha(m, n)}{\sum_{n'=1}^{M^{(s)}} \exp \psi_\alpha(m, n')} \quad [18.39]$$

Sigmoid attention

$$\alpha_{m \rightarrow n} = \sigma(\psi_\alpha(m, n)) \quad [18.40]$$

The attention α is then used to compute an **context vector** \mathbf{c}_m by taking a weighted average over the columns of \mathbf{Z} ,

$$\mathbf{c}_m = \sum_{n=1}^{M^{(s)}} \alpha_{m \rightarrow n} \mathbf{z}_n, \quad [18.41]$$

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):

$$\tilde{\mathbf{h}}_m^{(t)} = \tanh(\Theta_c[\mathbf{h}_m^{(t)}; \mathbf{c}_m]) \quad [18.42]$$

$$\mathbf{p}(w_{m+1}^{(t)} | \mathbf{w}_{1:m}^{(t)}, \mathbf{w}^{(s)}) \propto \exp \left(\beta_{w_{m+1}^{(t)}} \cdot \tilde{\mathbf{h}}_m^{(t)} \right). \quad [18.43]$$

Here the decoder state $\mathbf{h}_m^{(t)}$ is concatenated with the context vector, forming the input to compute a final output vector $\tilde{\mathbf{h}}_m^{(t)}$. The context vector can be incorporated into the decoder recurrence in a similar manner (Bahdanau et al., 2014).

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, \mathbf{z} , and the “queries” are state representations in the decoder network $\mathbf{h}^{(t)}$. 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:

$$\mathbf{z}_m^{(i)} = \sum_{n=1}^{M^{(s)}} \alpha_{m \rightarrow n}^{(i)} (\Theta_v \mathbf{h}_n^{(i-1)}) \quad [18.44]$$

$$\mathbf{h}_m^{(i)} = \Theta_2 \text{ReLU} \left(\Theta_1 \mathbf{z}_m^{(i)} + \mathbf{b}_1 \right) + \mathbf{b}_2. \quad [18.45]$$

9664 The equations state that for each token m , level i computes self-attention over the
 9665 entire source sentence: the keys, values, and queries are all projections of the vector $\mathbf{h}^{(i-1)}$.
 9666 The attention scores $\alpha_{m \rightarrow n}^{(i)}$ are computed using a scaled form of softmax attention,

$$\alpha_{m \rightarrow n} \propto \exp(\psi_\alpha(m, n)/M), \quad [18.46]$$

9667 where M is the length of the input. This encourages the attention to be more evenly
 9668 dispersed across the input. Self-attention is applied across multiple “heads”, each using
 9669 different projections of $\mathbf{h}^{(i-1)}$ to form the keys, values, and queries.

9670 The output of the self-attentional layer is the representation $\mathbf{z}_m^{(i)}$, which is then passed
 9671 through a two-layer feed-forward network, yielding the input to the next layer, $\mathbf{h}^{(i)}$. To
 9672 ensure that information about word order in the source is integrated into the model, the
 9673 encoder includes **positional encodings** of the index of each word in the source. These
 9674 encodings are vectors for each position $m \in \{1, 2, \dots, M\}$. The positional encodings are
 9675 concatenated with the word embeddings \mathbf{x}_m at the base layer of the model.⁶

9676 Convolutional neural networks (see § 3.4) have also been applied as encoders in neu-
 9677 ral machine translation. For each word $w_m^{(s)}$, a convolutional network computes a rep-
 9678 resentation $\mathbf{h}_m^{(s)}$ from the embeddings of the word and its neighbors. This procedure is
 9679 applied several times, creating a deep convolutional network. The recurrent decoder then
 9680 computes a set of attention weights over these convolutional representations, using the
 9681 decoder’s hidden state $\mathbf{h}^{(t)}$ as the queries. This attention vector is used to compute a
 9682 weighted average over the outputs of *another* convolutional neural network of the source,
 9683 yielding an averaged representation \mathbf{c}_m , which is then fed into the decoder. As with the
 9684 transformer, speed is the main advantage over recurrent encoding models; another sim-
 9685 ilarity is that word order information is approximated through the use of positional en-
 9686 codings. It seems likely that there are limitations to how well positional encodings can
 9687 account for word order and deeper linguistic structure. But for the moment, the com-

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).

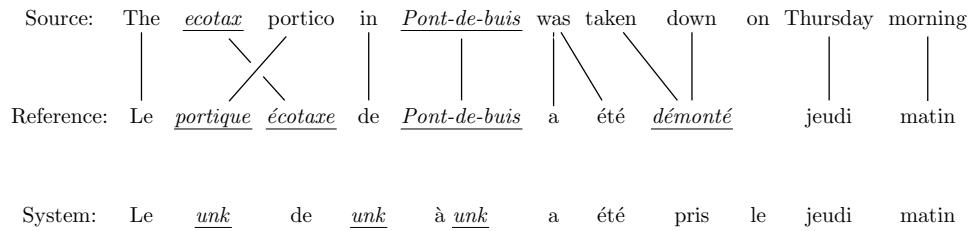


Figure 18.7: Translation with *unknown words*. The system outputs *unk* to indicate words that are outside its vocabulary. Figure adapted from Luong et al. (2015).

putational advantages of such approaches have put them on par with the best recurrent
translation models.⁷

18.3.3 Out-of-vocabulary words

Thus far, we have treated translation as a problem at the level of words or phrases. For words that do not appear in the training data, all such models will struggle. There are two main reasons for the presence of out-of-vocabulary (OOV) words:

- New proper nouns, such as family names or organizations, are constantly arising — particularly in the news domain. The same is true, to a lesser extent, for technical terminology. This issue is shown in Figure 18.7.
- In many languages, words have complex internal structure, known as **morphology**. An example is German, which uses compounding to form nouns like *Abwasserbehandlungsanlage* (*sewage water treatment plant*; example from Sennrich et al. (2016)). While compounds could in principle be addressed by better tokenization (see § 8.4), other morphological processes involve more complex transformations of subword units.

Cases such as names and technical terms can be handled in a postprocessing step. After first identifying alignments between unknown words in the source and target, we can look up each aligned source word in a dictionary, and choose a replacement (Luong et al., 2015). If the word does not appear in the dictionary, it is likely to be a proper noun, and can be copied directly from the source to the target. This approach can also be integrated directly into the translation model, rather than applying it as a postprocessing step (Jean et al., 2015).

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.

Words with complex internal structure can be handled by translating subword units rather than entire words. A popular technique for identifying subword units is **byte pair encoding** (BPE; Gage, 1994; Sennrich et al., 2016). The initial vocabulary is defined as the set of characters used in the text. The most common character bigram is then merged into a new symbol, and the vocabulary is updated. The merging operation is applied repeatedly, until the vocabulary reaches some maximum size. For example, given the dictionary $\{low, lowest, newer, wider\}$, the following sequence of mergers would be performed:⁸

$$r \blacksquare \rightarrow r\blacksquare \quad [18.47]$$

$$l o \rightarrow lo \quad [18.48]$$

$$lo w \rightarrow low \quad [18.49]$$

$$e r\blacksquare \rightarrow er\blacksquare. \quad [18.50]$$

- 9710 In the resulting vocabulary, the word *lowest* would be analyzed as four subword units:
 9711 $low+e+s+t$. Each subword unit is treated as a unique token for translation, in both the en-
 9712 coder (source side) and decoder (target side). BPE can be applied jointly to the union of the
 9713 source and target vocabularies, identifying subword units that appear in both languages.
 9714 For languages that have different scripts, such as English and Russian, **transliteration**
 9715 between the scripts should be applied first.⁹

9716 18.4 Decoding

Given a trained translation model, the decoding task is:

$$\hat{\mathbf{w}}^{(t)} = \underset{\mathbf{w} \in \mathcal{V}^*}{\operatorname{argmax}} \Psi(\mathbf{w}, \mathbf{w}^{(s)}), \quad [18.51]$$

- 9717 where $\mathbf{w}^{(t)}$ is a sequence of tokens from the vocabulary \mathcal{V} . As shown below, it is not
 9718 possible to efficiently obtain exact solutions to the decoding problem, for even minimally
 9719 effective models in either statistical or neural machine translation. Today's state-of-the-art
 9720 translation systems use **beam search** (see § 11.3.1.4), which is an incremental decoding al-
 9721 gorithm that maintains a small constant number of competitive hypotheses. Such greedy
 9722 approximations are reasonably effective in practice, and this may be in part because the
 9723 decoding objective is only loosely correlated with measures of translation quality, so that
 9724 exact optimization of [18.51] may not produce great translations.

⁸This example is from Sennrich et al. (2016). [todo: replace with an example in which there aren't so many ties.]

⁹Transliteration 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).

Decoding in neural machine translation is somewhat simpler than in phrase-based statistical machine translation.¹⁰ The scoring function Ψ is defined,

$$\Psi(\mathbf{w}^{(t)}, \mathbf{w}^{(s)}) = \sum_{m=1}^{M^{(t)}} \psi(w_m^{(t)}; \mathbf{w}_{1:m-1}^{(t)}, \mathbf{z}) \quad [18.52]$$

$$\psi(w^{(t)}; \mathbf{w}_{1:m-1}^{(t)}, \mathbf{z}) = \beta_{w_m^{(t)}} \cdot \mathbf{h}_m^{(t)} - \log \sum_{w \in \mathcal{V}} \exp(\beta_w \cdot \mathbf{h}_m^{(t)}), \quad [18.53]$$

where \mathbf{z} is the encoding of the source sentence $\mathbf{w}^{(s)}$, and $\mathbf{h}_m^{(t)}$ is a function of the encoding \mathbf{z} and the decoding history $\mathbf{w}_{1:m-1}^{(t)}$. This formulation subsumes the attentional translation model, where \mathbf{z} is a matrix encoding of the source, and $\mathbf{h}_m^{(t)}$ includes an attention-weighted sum over its columns.

Now consider the incremental decoding algorithm,

$$\hat{w}_m^{(t)} = \operatorname{argmax}_{w \in \mathcal{V}} \psi(w; \hat{\mathbf{w}}_{1:m-1}^{(t)}, \mathbf{z}), \quad \forall m \in \{1, 2, \dots\} \quad [18.54]$$

This algorithm selects the best target language word at position m , assuming that it has already generated the sequence $\hat{\mathbf{w}}_{1:m-1}^{(t)}$. (Termination can be handled by augmenting the vocabulary \mathcal{V} with a special end-of-sequence token, ■.) The incremental algorithm is likely to produce a suboptimal solution to the optimization problem defined in Equation 18.51, because selecting the highest-scoring word at position m can set the decoder on a “garden path,” in which there are no good choices at some later position $n > m$. We might hope for some dynamic programming solution, as in sequence labeling (§ 7.3). But the Viterbi algorithm and its relatives rely on a Markov decomposition of the objective function into a sum of local scores: for example, scores can consider locally adjacent tags (y_m, y_{m-1}) , but not the entire tagging history $y_{1:m}$. This decomposition is not applicable to recurrent neural networks, because the hidden state $\mathbf{h}_m^{(t)}$ is impacted by the entire history $\mathbf{w}_{1:m}^{(t)}$; this sensitivity to long-range context is precisely what makes recurrent neural networks so effective.¹¹ In fact, it can be shown that decoding from any recurrent neural network is NP-complete (Siegelmann and Sontag, 1995; Chen et al., 2018).

Beam search Beam search is a general technique for avoiding search errors when exhaustive search is impossible; it was first discussed in § 11.3.1.4. Beam search can be seen as a variant of the incremental decoding algorithm sketched in Equation 18.54, but at each step m , a set of K different hypotheses are kept on the beam. For each hypothesis

¹⁰For more on decoding in statistical translation, especially phrase-based models, see Koehn (2009).

¹¹Note 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.

9747 $k \in \{1, 2, \dots, K\}$, we compute both the current score $\sum_{m=1}^{M^{(t)}} \psi(w_{k,m}^{(t)}; \mathbf{w}_{k,1:m-1}^{(t)}, \mathbf{z})$ as well as
 9748 the current hidden state $\mathbf{h}_k^{(t)}$. At each step in the beam search, the K top-scoring children
 9749 of each hypothesis currently on the beam are “expanded”, and the beam is updated. For
 9750 a detailed description of beam search for RNN decoding, see Graves (2012).

9751 **Learning and search** Conventionally, the learning algorithm is trained to predict the
 9752 right token in the translation, conditioned on the translation history being correct. But
 9753 if decoding must be approximate, then we might do better by modifying the learning
 9754 algorithm to be robust to errors in the translation history. **Scheduled sampling** does this
 9755 by training on histories that sometimes come from the ground truth, and sometimes come
 9756 from the model’s own output (Bengio et al., 2015).¹² As training proceeds, the training
 9757 wheels come off: we increase the fraction of tokens that come from the model rather than
 9758 the ground truth. Another approach is to train on an objective that relates directly to beam
 9759 search performance (Wiseman et al., 2016). **Reinforcement learning** has also been applied
 9760 to decoding of RNN-based translation models, making it possible to directly optimize
 9761 translation metrics such as BLEU (Ranzato et al., 2016).

9762 18.5 Training towards the evaluation metric

9763 In likelihood-based training, the objective is the maximize the probability of a parallel
 9764 corpus. However, translations are not evaluated in terms of likelihood: metrics like BLEU
 9765 consider only the correctness of a single output translation, and not the range of prob-
 9766 abilities that the model assigns. It might therefore be better to train translation models
 9767 to achieve the highest BLEU score possible — to the extent that we believe BLEU mea-
 9768 sures translation quality. Unfortunately, BLEU and related metrics are not friendly for
 9769 optimization: they are discontinuous, non-differentiable functions of the parameters of
 9770 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,

$$\hat{\mathbf{w}}^{(t)} = \underset{\mathbf{w}^{(t)}}{\operatorname{argmax}} \Psi(\mathbf{w}^{(t)}, \mathbf{w}^{(s)}; \theta) \quad [18.55]$$

$$\min_{\theta} \Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(s)}) \quad [18.56]$$

9771 However, identifying the top-scoring translation $\hat{\mathbf{w}}^{(t)}$ is usually intractable, as described
 9772 in the previous section. In **minimum error-rate training (MERT)**, $\hat{\mathbf{w}}^{(t)}$ is selected from a

¹²Scheduled 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.

9773 set of candidate translations $\mathcal{Y}(\mathbf{w}^{(s)})$; this is typically a strict subset of all possible transla-
 9774 tions, so that it is only possible to optimize an approximation to the true error rate (Och
 9775 and Ney, 2003).

A further issue is that the objective function in Equation 18.56 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 define the **risk** as the expected error rate,

$$R(\theta) = E_{\hat{\mathbf{w}}^{(t)} | \mathbf{w}^{(s)}; \theta} [\Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)})] \quad [18.57]$$

$$= \sum_{\hat{\mathbf{w}}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})} p(\hat{\mathbf{w}}^{(t)} | \mathbf{w}^{(s)}) \times \Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)}). \quad [18.58]$$

9776 **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,

$$\tilde{p}(\mathbf{w}^{(t)}; \theta, \alpha) \propto \left(p(\mathbf{w}^{(t)} | \mathbf{w}^{(s)}; \theta) \right)^\alpha \quad [18.59]$$

$$\tilde{R}(\theta) = \sum_{\hat{\mathbf{w}}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})} \tilde{p}(\hat{\mathbf{w}}^{(t)} | \mathbf{w}^{(s)}; \alpha, \theta) \times \Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)}) \quad [18.60]$$

9777 where $\mathcal{Y}(\mathbf{w}^{(s)})$ is now the set of *all* possible translations for $\mathbf{w}^{(s)}$. (Exponentiating the
 9778 probabilities in this way is known as **annealing** (Smith and Eisner, 2006).) When $\alpha = 1$,
 9779 then $\tilde{R}(\theta) = R(\theta)$; when $\alpha = \infty$, then $\tilde{R}(\theta)$ is equivalent to the sum of the errors of the
 9780 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 $\sum_{\hat{\mathbf{w}}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})}$ 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),

$$\tilde{R}(\theta) \approx \frac{1}{Z} \sum_{k=1}^K \tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)}; \theta, \alpha) \times \Delta(\mathbf{w}_k^{(t)}, \mathbf{w}^{(t)}) \quad [18.61]$$

$$Z = \sum_{k=1}^K \tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)}; \theta, \alpha). \quad [18.62]$$

9781 Shen et al. (2016) report that performance plateaus at $K = 100$ for minimum risk training
 9782 of neural machine translation.

Uniform sampling over the set of all possible translations is undesirable, because most translations have very low probability. A solution from Monte Carlo estimation is **importance sampling**, in which we draw samples from a **proposal distribution** $q(\mathbf{w}^{(t)})$. This distribution can be set equal to the current translation model $p(\mathbf{w}^{(t)} | \mathbf{w}^{(s)}; \boldsymbol{\theta})$. Each sample is then weighted by an **importance score**, $\omega_k = \frac{\tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)})}{q(\mathbf{w}_k^{(t)})}$. The effect of this weighting is to correct for any mismatch between the proposal distribution q and the true distribution \tilde{p} . The risk can then be approximated as,

$$\mathbf{w}_k^{(t)} \sim q(\mathbf{w}^{(t)}) \quad [18.63]$$

$$\omega_k = \frac{\tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)})}{q(\mathbf{w}_k^{(t)})} \quad [18.64]$$

$$\tilde{R}(\boldsymbol{\theta}) \approx \frac{1}{\sum_{k=1}^K \omega_k} \sum_{k=1}^K \omega_k \times \Delta(\mathbf{w}_k^{(t)}, \mathbf{w}^{(t)}). \quad [18.65]$$

Importance sampling will generally give a more accurate approximation with a given number of samples. The only formal requirement is that the proposal assigns non-zero probability to every $\mathbf{w}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})$. For more on importance sampling and related methods, see Robert and Casella (2013).

Additional readings and software

A complete textbook on machine translation is available from Koehn (2009). While this book precedes recent work on neural translation, a more recent draft chapter on neural translation models is also available (Koehn, 2017). Neubig (2017) provides a comprehensive tutorial on neural machine translation, starting from first principles. The course notes from Cho (2015) are also useful.

Several neural machine translation systems are available, in connection with each of the major neural computing libraries: `lamtram` is an implementation of neural machine translation in the `dynet` (Neubig et al., 2017); `OpenNMT` (Klein et al., 2017) is an implementation in `PyTorch`; `tensor2tensor` is an implementation of several of the Google translation models in `tensorflow` (Abadi et al., 2016).

Literary translation is especially challenging, even for expert human translators. Mes-sud (2014) describes some of these issues in her review of an English translation of *L'étranger*, the 1942 French novel by Albert Camus.¹³ She compares the new translation by Sandra Smith against earlier translations by Stuart Gilbert and Matthew Ward, focusing on the difficulties presented by a single word in the first sentence:

¹³The book review is currently available online at <http://www.nybooks.com/articles/2014/06/05/camus-new-letranger/>.

Then, too, Smith has reconsidered the book's famous opening. Camus's original is deceptively simple: "*Aujourd'hui, maman est morte.*" Gilbert influenced generations by offering us "Mother died today"—inscribing in Meursault [the narrator] from the outset a formality that could be construed as heartlessness. But *maman*, after all, is intimate and affectionate, a child's name for his mother. Matthew Ward concluded that it was essentially untranslatable ("mom" or "mummy" being not quite apt), and left it in the original French: "Maman died today." There is a clear logic in this choice; but as Smith has explained, in an interview in *The Guardian*, *maman* "didn't really tell the reader anything about the connotation." She, instead, has translated the sentence as "My mother died today."

I chose "My mother" because I thought about how someone would tell another person that his mother had died. Meursault is speaking to the reader directly. "My mother died today" seemed to me the way it would work, and also implied the closeness of "maman" you get in the French.

Elsewhere in the book, she has translated *maman* as "mama"—again, striving to come as close as possible to an actual, colloquial word that will carry the same connotations as *maman* does in French.

The passage is a useful reminder that while the quality of machine translation has improved dramatically in recent years, expert human translations draw on considerations that are beyond the ken of any known computational approach.

Exercises

1. Give a synchronized derivation (§ 18.2.4) for the Spanish-English translation,

(18.4) *El pez enojado atacado.*
 The fish angry attacked.
 The angry fish attacked.

As above, the second line shows a word-for-word gloss, and the third line shows the desired translation. Use the synchronized production rule in [18.24], and design the other production rules necessary to derive this sentence pair. You may derive (*atacado*, *attacked*) directly from VP.

9833 Chapter 19

9834 Text generation

9835 Many of the most interesting problems in natural language processing involve language
9836 as the output. The previous chapter described the specific case of machine translation,
9837 but there are many other applications, ranging from summarization of research articles,
9838 to automated journalism, to dialogue systems. This chapter emphasizes three main sce-
9839 narios: data-to-text, in which text is generated to explain or describe a structured record
9840 or unstructured perceptual input; text-to-text, which typically involves fusing informa-
9841 tion from multiple linguistic sources into a single coherent summary; and dialogue, in
9842 which text is generated as part of an interactive conversation with one or more human
9843 participants.

9844 19.1 Data-to-text generation

9845 Data-to-text generation subsumes a number of scenarios. The form of the data can range
9846 from a structured record, such as the description of a weather forecast (as shown in
9847 Figure 19.1), to unstructured perceptual data, such as a raw image or video. Similarly,
9848 the form of the output can range from a single sentence, such as an image caption, to a
9849 multi-paragraph argument. Nonetheless, all such systems share some of the same chal-
9850 lenges (Reiter and Dale, 2000):

- 9851 • determining what parts of the data to describe in text;
- 9852 • planning a presentation of this information;
- 9853 • **lexicalizing** the data into words and phrases;
- 9854 • organizing words and phrases into well-formed sentences and paragraphs.

9855 The earlier stages of this process are sometimes called **content selection** and **text plan-**
9856 **ning**; the later stages are often called **surface realization**.

Database:	Temperature			Cloud Sky Cover		
	time	min	mean	max	time	percent (%)
	06:00-21:00	9	15	21	06:00-09:00	25-50
	09:00-12:00				09:00-12:00	50-75
Wind Speed			Wind Direction			
			time	mode		
	06:00-21:00	15	20	30	06:00-21:00	S

Text: Cloudy, with temperatures between 10 and 20 degrees. South wind around 20 mph.

Figure 19.1: An example input-output pair for the task of generating text descriptions of weather forecasts (Konstas and Lapata, 2013). [todo: permission]

9857 Early systems for data-to-text generation were modular, with separate software components for each task. Artificial intelligence **planning** algorithms can play a role in both
 9858 the high-level information structure and the organization of individual sentences, ensuring
 9859 that communicative goals are met (McKeown, 1992; Moore and Paris, 1993). Surface
 9860 realization can be performed by grammars or templates, which link specific types of data
 9861 to candidate words and phrases. A simple example is offered by Wiseman et al. (2017),
 9862 who built a template-based system for generating description of basketball games:
 9863

- 9864 (19.1) The <team1>(<wins1>-<losses1>) defeated the <team2>(<wins2>-<losses2>),
 9865 <pts1>-<pts2>.
 9866 The New York Knicks (45-5) defeated the Boston Celtics (11-38), 115-79.

9867 For more complex cases, it may be necessary to apply morphological inflections such as
 9868 pluralization and tense marking — even in the simple example above, languages such
 9869 as Russian would require case marking suffixes for the team names. Such inflections can
 9870 be applied as a postprocessing step. Another difficult challenge for surface realization is
 9871 the generation of varied **referring expressions** (e.g., *The Knicks*, *New York*, *they*), which is
 9872 critical to avoid repetition. As discussed in § 16.2.1, the form of referring expressions is
 9873 constrained by the discourse and information structure.

9874 An example at the intersection of rule-based and statistical techniques is the Nitrogen
 9875 system (Langkilde and Knight, 1998). The input to Nitrogen is an abstract meaning rep-
 9876 resentation (AMR; see § 13.3) of semantic content to be expressed in a single sentence. In
 9877 data-to-text scenarios, the abstract meaning representation is the output of a higher-level
 9878 text planning stage. A set of rules then converts the abstract meaning representation into
 9879 various sentence plans, which may differ in both the high-level structure (e.g., active ver-
 9880 sus passive voice) as well as the low-level details (e.g., word and phrase choice). Some
 9881 examples are shown in Figure 19.2. To control the combinatorial explosion in the number

```
(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).

of possible realizations for any given meaning, the sentence plans are unified into a single finite-state acceptor, in which each word is an arc (see § 9.1.1). A bigram 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. 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, it would be helpful to decompose the scoring function Ψ into subcomponents. This would be possible if we were given an **alignment**, specifying which element of \mathbf{y} is expressed in each part of \mathbf{w} (Angeli et al., 2010):

$$\Psi(\mathbf{w}, \mathbf{y}; \theta) = \sum_{m=1}^M \psi_{w,y}(\mathbf{w}_m, \mathbf{y}_{z_m}) + \psi_z(z_m, z_{m-1}), \quad [19.2]$$

where z_m indicates the record aligned to word m . For example, in Figure 19.1, z_1 might specify that the word *cloudy* is aligned to the record `cloud-sky-cover:percent`. The

9904 score for this alignment would then be given by the weight on features such as

$$(cloudy, \text{cloud-sky-cover:percent}), \quad [19.3]$$

9905 which could be learned from labeled data $\{(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})\}_{i=1}^N$. The function ψ_z can learn
 9906 to assign higher scores to alignments that are coherent, referring to the same records in
 9907 adjacent parts of the text.¹

9908 Several datasets include structured records and natural language text (Barzilay and
 9909 McKeown, 2005; Chen and Mooney, 2008; Liang and Klein, 2009), but the alignments
 9910 between text and records are usually not available.² One solution is to model the problem
 9911 probabilistically, treating the alignment as a latent variable (Liang et al., 2009; Konstas
 9912 and Lapata, 2013). The model can then be estimated using expectation maximization (see
 9913 chapter 5).

9914 19.1.2 Neural data-to-text generation

9915 The **encoder-decoder model** and **neural attention** were introduced in § 18.3 as methods
 9916 for neural machine translation. These models can also be applied to data-to-text gener-
 9917 ation, with the data acting as the source language (Mei et al., 2016). In neural machine
 9918 translation, the attention mechanism linked words in the source to words in the target;
 9919 in data-to-text generation, the attention mechanism can link each part of the generated
 9920 text back to a record in the data. The biggest departure from translation is in the encoder,
 9921 which depends on the form of the data.

9922 19.1.2.1 Data encoders

9923 In some types of structured records, all values are drawn from discrete sets. For example,
 9924 the birthplace of an individual is drawn from a discrete set of possible locations; the diag-
 9925 nosis and treatment of a patient are drawn from an exhaustive list of clinical codes (John-
 9926 son et al., 2016). In such cases, it is possible to learn vector embeddings for each field
 9927 and possible value: for example, a vector embedding for the field BIRTHPLACE, and an-
 9928 other embedding for the value Berkeley_California (Bordes et al., 2011). The table of
 9929 such embeddings then serves as the encoding of a structured record (He et al., 2017). It
 9930 is also possible to compress the table into a vector representation, by **pooling** across the
 9931 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.

²One exception is a dataset of records and summaries from American football games, containing annotations of alignments between sentences and records (Snyder and Barzilay, 2007).

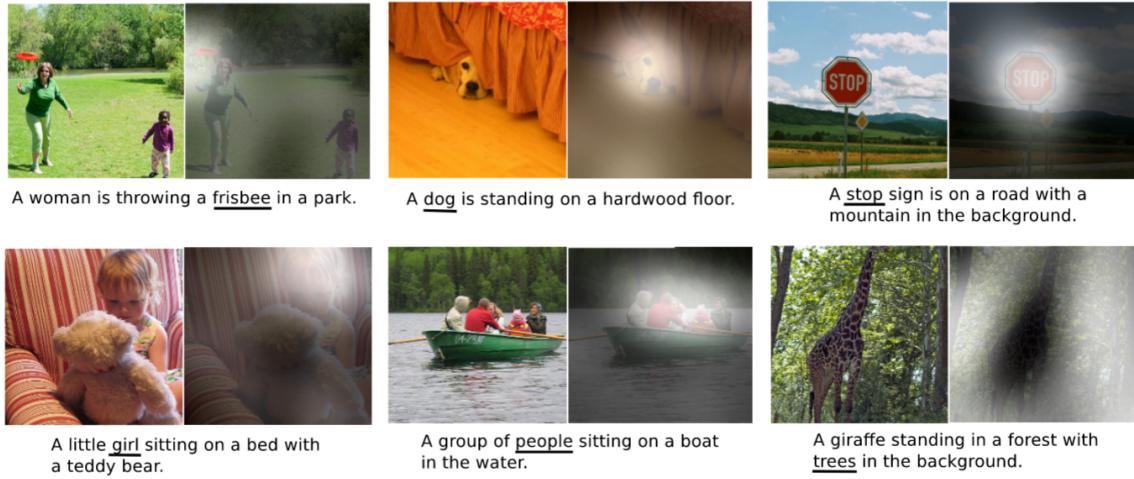


Figure 19.3: Examples of the image captioning task, with attention masks shown for each of the underlined words. From Xu et al. (2015). [todo: permission]

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,

```
PASS(arg1 = purple6,arg2 = purple3)
KICK(arg1 = purple3)
BADPASS(arg1 = purple3,arg2 = pink9),
```

describes a sequence of events in a robot soccer match (Mei et al., 2016). 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), e.g.,

$$\mathbf{X} = [\mathbf{u}_{\text{PASS}}, \mathbf{u}_{\text{arg1}}, \mathbf{u}_{\text{purple6}}, \mathbf{u}_{\text{arg2}}, \mathbf{u}_{\text{purple3}}]. \quad [19.4]$$

This encoding can then act as the input layer for a recurrent neural network, yielding a sequence of vector representations $\{\mathbf{z}_r\}_{r=1}^R$, where r indexes over records. Interestingly, Mei et al. (2016) show that this sequence-based approach is effective even in cases where there is no natural ordering over the records, such as the weather data in Figure 19.1.

Images Another flavor of data-to-text generation is the generation of text captions for 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 $320 \times 240 \times 3$ intensity values. The dominant approach to image classification is to encode images as vectors using a combination of convolution and pooling (Krizhevsky et al.,

9944 2012). chapter 3 explains how to use convolutional networks for text; for images, convolution
 9945 is applied across the vertical, horizontal, and color dimensions. By pooling the results
 9946 of successive convolutions, the image is converted to a vector representation, which can
 9947 then be fed directly into the decoder as the initial state (Vinyals et al., 2015), just as in the
 9948 sequence-to-sequence translation model (see § 18.3). Alternatively, one can apply a set of
 9949 convolutional networks, yielding vector representations for different parts of the image,
 9950 which can then be combined using neural attention (Xu et al., 2015).

9951 **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 , we can compute an attention vector over the data using the same technique described in § 18.3.1. When generating word m of the output, a softmax attention mechanism computes the weighted average \mathbf{c}_m ,

$$\psi_\alpha(m, r) = \beta_\alpha \cdot f(\Theta_\alpha[\mathbf{h}_m; \mathbf{z}_r]) \quad [19.5]$$

$$\boldsymbol{\alpha}_m = \text{SoftMax}([\psi_\alpha(m, 1), \psi_\alpha(m, 2), \dots, \psi_\alpha(m, R)]) \quad [19.6]$$

$$\mathbf{c}_m = \sum_{r=1}^R \alpha_{m \rightarrow r} \mathbf{z}_r, \quad [19.7]$$

9952 where f is an elementwise nonlinearity such as tanh or ReLU (see § 3.2.1). The weighted
 9953 average \mathbf{c}_m can then be included in the recurrent update to the decoder state, or in the
 9954 emission probabilities, as described in § 18.3.1. Figure 19.4 shows the attention to compo-
 9955 nents of a weather record, while generating the text shown on the x -axis.

9956 Adapting this architecture to image captioning is straightforward: rather than com-
 9957 putting attention over a set of records, we can apply convolutional neural networks to a
 9958 set of image locations, and represent the output at each location ℓ with a vector \mathbf{z}_ℓ . Attention
 9959 can then be computed over the image locations, as shown in the right panels of each
 9960 pair of images in Figure 19.3.

9961 Various modifications to this basic mechanism have been proposed. In **coarse-to-fine**
 9962 **attention** (Mei et al., 2016), each record receives a global attention $a_r \in [0, 1]$, which is in-
 9963 dependent of the decoder state. This global attention, which represents the overall impor-
 9964 tance of the record, is multiplied with the decoder-based attention scores, before comput-
 9965 ing the final normalized attentions. In **structured attention**, the attention vector $\boldsymbol{\alpha}_{m \rightarrow \cdot}$ can
 9966 include structural biases, which can favor assigning higher attention values to contiguous
 9967 segments or to dependency subtrees (Kim et al., 2017). Structured attention vectors can
 9968 be computed by running the forward-backward algorithm to obtain marginal attention
 9969 probabilities (see § 7.5.3.3). Because each step in the forward-backward algorithm is dif-
 9970 ferentiable, it can be encoded in a computation graph, and end-to-end learning remains
 9971 possible.

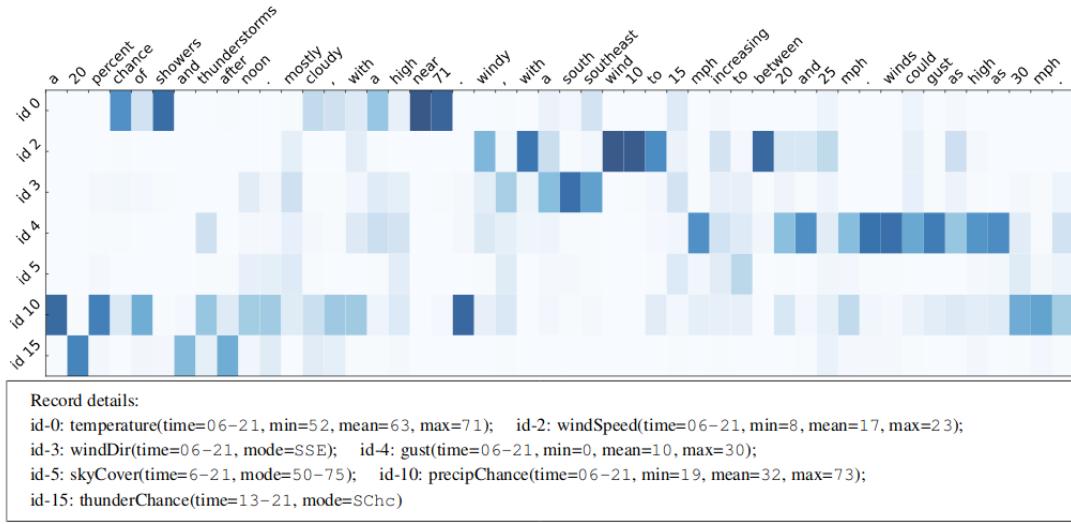


Figure 19.4: Neural attention in text generation. Figure from Mei et al. (2016).

9972 19.1.2.3 Decoder

9973 Given the encoding, the decoder can function just as in neural machine translation (see
 9974 § 18.3.1), using the attention-weighted encoder representation in the decoder recurrence
 9975 and/or output computation. As in machine translation, beam search can help the search
 9976 algorithm to explore multiple hypotheses (Lebret et al., 2016).

In many applications, it can be important to generate 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,

$$p(w^{(t)} | w_{1:m-1}^{(t)}, \mathbf{Z}, s_m) = \begin{cases} \text{SoftMax}(\beta_{w^{(t)}} \cdot h_{m-1}^{(t)}), & s_m = \text{gen} \\ \sum_{r=1}^R \delta(w_r^{(s)} = w^{(t)}) \times \alpha_{m \rightarrow r}, & s_m = \text{copy}, \end{cases} \quad [19.8]$$

9977 where $\delta(w_r^{(s)} = w^{(t)})$ is an indicator function, taking the value 1 iff the text of the record
 9978 $w_r^{(s)}$ is identical to the target word $w^{(t)}$. The probability of copying record r from the
 9979 source is $s_m \times \alpha_{m \rightarrow r}$, the product of the copy probability by the local attention. Note
 9980 that in this model, the attention weights α_m are computed from the previous decoder state

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.

9981 \mathbf{h}_{m-1} . The computation graph therefore remains a feedforward network, with recurrent
 9982 paths such as $\mathbf{h}_{m-1}^{(t)} \rightarrow \alpha_m \rightarrow w_m^{(t)} \rightarrow \mathbf{h}_m^{(t)}$.

9983 To facilitate end-to-end training, the switching variable s_m can be represented by a
 9984 gate π_m , which is computed from a two-layer feedforward network, whose input consists
 9985 of the concatenation of the decoder state $\mathbf{h}_{m-1}^{(t)}$ and the attention-weighted representation
 9986 of the data, $\mathbf{c}_m = \sum_{r=1}^R \alpha_{m \rightarrow r} \mathbf{z}_r$,

$$\pi_m = \sigma(\Theta^{(2)} f(\Theta^{(1)}[\mathbf{h}_{m-1}^{(t)}; \mathbf{c}_m])). \quad [19.9]$$

The full generative probability at token m is then,

$$p(w^{(t)} | \mathbf{w}_{1:m}^{(t)}, \mathbf{Z}) = \pi_m \times \underbrace{\frac{\exp \beta_{w^{(t)}} \cdot \mathbf{h}_{m-1}^{(t)}}{\sum_{j=1}^V \exp \beta_j \cdot \mathbf{h}_{m-1}^{(t)}}}_{\text{generate}} + (1 - \pi_m) \times \underbrace{\sum_{r=1}^R \delta(w_r^{(s)} = w^{(t)}) \times \alpha_{m \rightarrow r}}_{\text{copy}}. \quad [19.10]$$

9987 19.2 Text-to-text generation

9988 Text-to-text generation includes problems such as summarization and simplification:

- 9989 • reading a novel and outputting a paragraph-long summary of the plot;⁴
- 9990 • reading a set of blog posts about a contemporary political issue, and outputting a
 9991 bullet list of the various issues and perspectives;
- 9992 • reading a technical research article about the long-term health consequences of drink-
 9993 ing kombucha, and outputting a summary of the article in language that non-experts
 9994 can understand.

9995 These problems can be approached in two ways: through the encoder-decoder architec-
 9996 ture discussed in the previous section, or by operating directly on the input text.

9997 19.2.1 Neural abstractive summarization

9998 **Sentence summarization** is the task of shortening a sentence while preserving its mean-
 9999 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.

- 10000 (19.2) The documentation is typical of Epson quality: excellent.
 10001 Documentation is excellent.
 10002
 10003 (19.3) Russian defense minister Ivanov called sunday for the creation of a joint front for
 10004 combating global terrorism.
 10005 Russia calls for joint front against terrorism.
 10006
- 10007 Sentence summarization is closely related to **sentence compression**, in which the sum-
 10008 mary is produced by deleting words or phrases from the original (Clarke and Lapata,
 10009 2008). But as shown in (19.3), a sentence summary can also introduce new words, such as
 10010 *against*, which replaces the phrase *for combatting*.

10011 Sentence summarization can be treated as a machine translation problem, using the at-
 10012 tentional encoder-decoder translation model discussed in § 18.3.1 (Rush et al., 2015). The
 10013 longer sentence is encoded into a sequence of vectors, one for each token. The decoder
 10014 then computes attention over these vectors when updating its own recurrent state. As
 10015 with data-to-text generation, it can be useful to augment the encoder-decoder model with
 10016 the ability to copy words directly from the source. Rush et al. (2015) train this model by
 10017 building four million sentence pairs from news articles. In each pair, the longer sentence is
 10018 the first sentence of the article, and the summary is the article headline. Sentence summa-
 10019 rization can also be trained in a semi-supervised fashion, using a probabilistic formulation
 10020 of the encoder-decoder model called a **variational autoencoder** (Miao and Blunsom, 2016,
 10021 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,

$$\ell_m = \sum_{n=1}^{M^{(s)}} \min(\alpha_{m \rightarrow n}, t_{m \rightarrow n}). \quad [19.12]$$

- 10022 This loss will be low if $\alpha_{m \rightarrow \cdot}$ assigns little attention to words that already have large
 10023 values in $t_{m \rightarrow \cdot}$. Coverage loss is similar to the concept of **marginal relevance**, in which
 10024 the reward for adding new content is proportional to the extent to which it increases

10025 the overall amount of information conveyed by the summary (Carbonell and Goldstein,
 10026 1998).

10027 **19.2.2 Sentence fusion for multi-document summarization**

10028 In **multi-document summarization**, the goal is to produce a summary that covers the
 10029 content of several documents (McKeown et al., 2002). One approach to this challenging
 10030 problem is to identify sentences across multiple documents that relate to a single theme,
 10031 and then to fuse them into a single sentence (Barzilay and McKeown, 2005). As an exam-
 10032 ple, consider the following two sentences (McKeown et al., 2010):

- 10033 (19.4) Palin actually turned against the bridge project only after it became a national
 10034 symbol of wasteful spending.
- 10035 (19.5) Ms. Palin supported the bridge project while running for governor, and aban-
 10036 doned it after it became a national scandal.

10037 An *intersection* preserves only the content that is present in both sentences:

- 10038 (19.6) Palin turned against the bridge project after it became a national scandal.

10039 A *union* includes information from both sentences:

- 10040 (19.7) Ms. Palin supported the bridge project while running for governor, but turned
 10041 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 § 13.2.2.1),

$$\max_{\mathbf{y}} \sum_{i,j,r} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) \times y_{i,j,r} \quad [19.13]$$

$$\text{s.t. } \mathbf{y} \in \mathcal{C}, \quad [19.14]$$

10042 where the variable $y_{i,j,r} \in \{0, 1\}$ indicates whether there is an edge from i to j of type r ,
 10043 and the score of this edge is $\psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})$. As usual, \mathbf{w} is the list of words in the graph,
 10044 and $\boldsymbol{\theta}$ is a vector of parameters. This score reflects the “importance” of the modifier to the
 10045 overall meaning: in intersective fusion, this score indicates the extent to which the content
 10046 in this edge is expressed in all sentences; in union fusion, the score indicates whether the
 10047 content in the edge is expressed in any sentence.

10048 The constraint set \mathcal{C} ensures that y forms a valid dependency graph, and can also
 10049 impose additional linguistic constraints: for example, ensuring that coordinated nouns
 10050 are sufficiently similar. The resulting tree must then be **linearized** into a sentence. This is
 10051 typically done by generating a set of candidate linearizations, and choosing the one with
 10052 the highest score under a language model (Langkilde and Knight, 1998; Song et al., 2016).

10053 [todo: needs figure]

10054 19.3 Dialogue

10055 **Dialogue systems** are capable of conversing with a human interlocutor, often to perform
 10056 some task (Grosz, 1979), but sometimes just to chat (Weizenbaum, 1966). While research
 10057 on dialogue systems goes back several decades (Carbonell, 1970; Winograd, 1972), com-
 10058 mercial systems such as Alexa and Siri have recently brought automated dialogue into
 10059 widespread use. Nonetheless, there is a significant gap between research and practice:
 10060 many practical dialogue systems remain scripted and inflexible, while research systems
 10061 emphasize abstractive text generation, “on-the-fly” decision making, and probabilistic
 10062 reasoning about the user’s intentions.

10063 19.3.1 Finite-state and agenda-based dialogue systems

10064 Finite-state automata were introduced in chapter 9 as a formal model of computation,
 10065 in which string inputs and outputs are linked to transitions between a finite number of
 10066 discrete states. This model naturally fits simple task-oriented dialogues, such as the one
 10067 shown in the left panel of Figure 19.5. This (somewhat frustrating) dialogue can be repre-
 10068 sented with a finite-state transducer, as shown in the right panel of the figure. The accept-
 10069 ing state is reached only when the two needed pieces of information are provided, and the
 10070 human user confirms that the order is correct. In this simple scenario, the TOPPING and
 10071 ADDRESS are the two **slots** associated with the activity of ordering a pizza, which is called
 10072 a **frame**. Frame representations can be hierarchical: for example, an ADDRESS could have
 10073 slots of its own, such as STREET and CITY.

10074 In the example dialogue in Figure 19.5, the user provides the precise inputs that are
 10075 needed in each turn (e.g., *anchovies*; *the College of Computing building*). Some users may
 10076 prefer to communicate more naturally, with phrases like *I’d, uh, like some anchovies please*.
 10077 One approach to handling such utterances is to design a grammar, in which specific non-
 10078 terminal are defined for the semantic **slots**, such as TOPPING and LOCATION. However,
 10079 context-free parsing of unconstrained speech input is challenging. Alternative approaches
 10080 include BIO-style sequence labeling (see § 8.3), which can be driven by an underlying
 10081 bi-directional recurrent neural network, similar to recurrent approaches to semantic role
 10082 labeling described in § 13.2.3. An example of BIO-style labeling for this task is:

- (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?
 ...

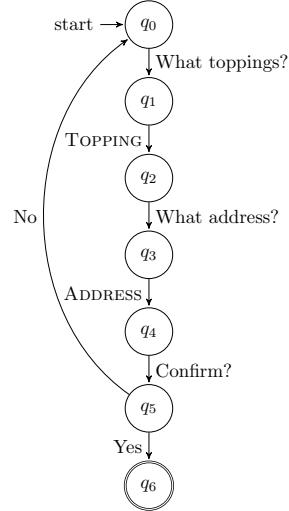


Figure 19.5: An example dialogue and the associated finite-state model. In the finite-state model, SMALL CAPS indicates that the user must provide information of this type in their answer.

- 10083 (19.9) *I'd like anchovies , and please bring it to the College of Computing*
 O O B-TOPPING O O O O O B-ADDR I-ADDR I-ADDR I-ADDR
 10084 *Building .*
 I-ADDR O

10085 The system illustrated in Figure 19.5 would not be capable of handling such an input:
 10086 it forces the user to provide the topping first, and then the location. In this sense, the
 10087 **initiative** is driven completely by the system. **Agenda-based dialogue systems** extend
 10088 finite-state architectures by attempting to recognize all slots that are filled by the user's
 10089 reply, thereby handling examples such as (19.9). The system then dynamically poses ad-
 10090 ditional questions, until the frame is complete (Bobrow et al., 1977; Allen et al., 1995;
 10091 Rudnicky and Xu, 1999). Such systems are said to be **mixed-initiative**, because both the
 10092 user and the system can drive the direction of the dialogue.

10093 19.3.2 Markov decision processes

10094 The task of dynamically selecting the next move in a conversation is known as **dialogue**
 10095 **management**. This problem can be framed as a **Markov decision process**, which is a
 10096 theoretical model that includes a discrete set of states, a discrete set of actions, a function
 10097 that computes the probability of transitions between states, and a function that computes
 10098 the cost or reward of action-state pairs. Let's see how each of these elements pertains to
 10099 the pizza ordering dialogue system.

- 10100 • Each state is a tuple of information about whether the topping and address are
 10101 known, and whether the order has been confirmed. For example,

$$(KNOWN\ TOPPING,\ UNKNOWN\ ADDRESS,\ NOT\ CONFIRMED) \quad [19.15]$$

10102 is a possible state. Any state in which the pizza order is confirmed is a terminal
 10103 state, and the Markov decision process stops after entering such a state.

- 10104 • The set of actions includes querying for the topping, querying for the address, and
 10105 requesting confirmation. Each action induces a probability distribution over states,
 10106 $p(s_t | a_t, s_{t-1})$. For example, requesting confirmation of the order is not likely to
 10107 result in a transition to the terminal state if the topping is not yet known. This
 10108 probability distribution over state transitions may be learned from data, or it may
 10109 be specified in advance.
- 10110 • Each state-action-state tuple earns a reward, $r_a(s_t, s_{t+1})$. In the context of the pizza
 10111 ordering system, a simple reward function would be,

$$r_a(s_t, s_{t+1}) = \begin{cases} 0, & a = \text{CONFIRM}, s_{t+1} = (*, *, \text{CONFIRMED}) \\ -10, & a = \text{CONFIRM}, s_{t+1} = (*, *, \text{NOT CONFIRMED}) \\ -1, & a \neq \text{CONFIRM} \end{cases} \quad [19.16]$$

10112 This function assigns zero reward for successful transitions to the terminal state,
 10113 a large negative reward to a rejected request for confirmation, and a small negative
 10114 reward for every other type of action. The system is therefore rewarded for reaching
 10115 the terminal state in few steps.

10116 In a Markov decision process, a **policy** is a function $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maps from states to
 10117 actions (see § 15.2.4.3). The value of a policy is the expected sum of discounted rewards,
 10118 $E_\pi[\sum_{t=1}^T \gamma^t r_{a_t}(s_t, s_{t+1})]$, where γ is the discount factor, $\gamma \in [0, 1)$. Discounting has the
 10119 effect of emphasizing rewards that can be obtained immediately over less certain rewards
 10120 in the distant future.

10121 An optimal policy can be obtained by dynamic programming, by iteratively updating
 10122 the **value function** $V(s)$, which is the expected cumulative rewards from s under the
 10123 optimal action a ,

$$V(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s' | s, a)[r_a(s, s') + \gamma V(s')]. \quad [19.17]$$

10124 Note that $V(s)$ is computed in terms of $V(s')$ for all states $s' \in \mathcal{S}$. A series of iterative up-
 10125 dates to the value function will eventually converge to a stationary point. This algorithm
 10126 is known as **value iteration**. Given the converged value function $V(s)$, the optimal action
 10127 at each state is then simply,

$$\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s' | s, a)[r_a(s, s') + \gamma V(s')]. \quad [19.18]$$

10128 Value iteration and related algorithms are described in detail by Sutton and Barto (1998).
 10129 Applications to dialogue systems are discussed by Levin et al. (1998) and Walker (2000).

10130 The Markov decision process framework assumes that the current state of the dialogue
 10131 is known. But in reality, the system may misinterpret the user’s statements — for example,
 10132 believing that a specification of the delivery location (PEACHTREE) is in fact a specification
 10133 of the topping (PEACHES). In a **partially observable Markov decision process (POMDP)**,
 10134 the system receives an *observation* o , which is probabilistically conditioned on the state,
 10135 $p(o | s)$. It must therefore maintain a distribution of beliefs about which state it is in, with
 10136 $q_t(s)$ indicating the degree of belief that the dialogue is in state s at time t . The POMDP
 10137 formulation can help to make dialogue systems more robust to errors, particularly in the
 10138 context of spoken language dialogues, where the speech itself may be misrecognized (Roy
 10139 et al., 2000; Williams and Young, 2007). However, finding the optimal policy in a POMDP
 10140 is computationally intractable, requiring additional approximations.

10141 **19.3.3 Neural chatbots**

10142 Chatting is a lot easier when you don’t need to get anything done. **Chatbots** simply try
 10143 to parry the user’s input with an appropriate response to keep the conversation going.
 10144 They can be built from the encoder-decoder architecture discussed in § 18.3 and § 19.1.2:
 10145 the encoder converts the user’s input into a vector, and the decoder produces a sequence
 10146 of words as a response. For example, Shang et al. (2015) apply the attentional encoder-
 10147 decoder translation model, training on a dataset of posts and responses from the Chinese
 10148 microblogging platform Sina Weibo.⁵ This approach is capable of generating replies that
 10149 relate thematically to the input, as shown in the following examples:⁶

- 10150 (19.10) High fever attacks me every New Year’s day.
 10151 Get well soon and stay healthy!
- 10152 (19.11) I gain one more year. Grateful to my group, so happy.
 10153 Getting old now. Time has no mercy.

10154 While encoder-decoder models can generate responses that make sense in the con-
 10155 text of the immediately preceding turn, they struggle to maintain coherence over longer
 10156 conversations. One solution is to model the dialogue context recurrently. This creates
 10157 a **hierarchical recurrent network**, including both word-level and turn-level recurrences.
 10158 The turn-level hidden state is then used as additional context in the decoder (Serban et al.,
 10159 2016), as shown in Figure 19.6.

⁵Twitter is also frequently used for construction of dialogue datasets (Ritter et al., 2011; Sordoni et al., 2015). Another source is technical support chat logs from the Ubuntu linux distribution (Uthus and Aha, 2013; Lowe et al., 2015).

⁶All examples are translated from Chinese by Shang et al. (2015).

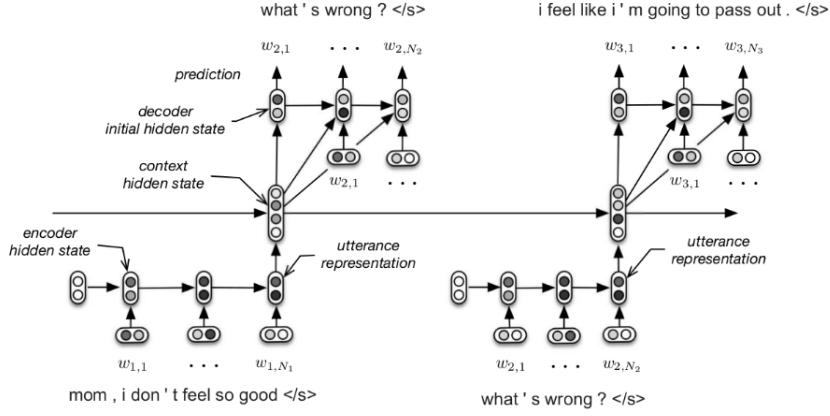


Figure 19.6: A hierarchical recurrent neural network for dialogue, with recurrence over both words and turns (Serban et al., 2016). [todo: redo]

10160 An important open question is how to integrate the encoder-decoder architecture into
 10161 task-oriented dialogue systems. An advantage of chatbots is that they can be trained end-
 10162 to-end: the user’s turn is analyzed by the encoder, and the system output is generated
 10163 by the decoder. This architecture can be trained by log-likelihood using backpropaga-
 10164 tion (e.g., Sordoni et al., 2015; Serban et al., 2016), or by more elaborate objectives, using
 10165 reinforcement learning (Li et al., 2016). In contrast, the task-oriented dialogue systems
 10166 described in § 19.3.1 typically involve a set of specialized modules: one for recognizing
 10167 the user input, another for deciding what action to take, and a third for arranging the text
 10168 of the system output.

10169 Recurrent neural network decoders can be integrated into Markov Decision Process
 10170 dialogue systems, by conditioning the decoder on a representation of the information
 10171 that is to be expressed in each turn (Wen et al., 2015). Specifically, the long short-term
 10172 memory (LSTM; § 6.3) architecture is augmented so that the memory cell at time m takes
 10173 an additional input d_m , which is a representation of the slots and values to be expressed
 10174 in the next turn. However, this approach still relies on additional modules to recognize
 10175 the user’s utterance and to plan the overall arc of the dialogue.

10176 Another promising direction is to create embeddings for the elements in the domain:
 10177 for example, the slots in a record and the entities that can fill them. The encoder then
 10178 encodes not only the words of the user’s input, but the embeddings of the elements that
 10179 the user mentions. Similarly, the decoder is endowed with the ability to refer to specific
 10180 elements in the knowledge base. He et al. (2017) show that such a method can learn to
 10181 play a collaborative dialogue game, in which both players are given a list of entities and
 10182 their properties, and the goal is to find an entity that is on both players’ lists.

(c) Jacob Eisenstein 2018. Work in progress.

10183 Further reading

10184 Gatt and Krahmer (2018) provide a comprehensive recent survey on text generation. For
10185 a book-length treatment of earlier work, see Reiter and Dale (2000). For a survey on image
10186 captioning, see Bernardi et al. (2016); for a survey of pre-neural approaches to dialogue
10187 systems, see Rieser and Lemon (2011). **Dialogue acts** were introduced in § 8.6 as a labeling
10188 scheme for human-human dialogues; they also play a critical role in task-based dialogue
10189 systems (Allen et al., 1996, e.g.). The incorporation of theoretical models of dialogue into
10190 computational systems is reviewed by Jurafsky and Martin (2009, chapter 24).

10191 While this chapter has focused on the informative dimension of text generation, another
10192 line of research aims to generate text with configurable stylistic properties (Walker
10193 et al., 1997; Mairesse and Walker, 2011; Fidler and Goldberg, 2017; Hu et al., 2017). This
10194 chapter also does not address the generation of creative text such as narratives (Riedl and
10195 Young, 2010), jokes (Ritchie, 2001), poems (Colton et al., 2012), and song lyrics (Gonçalo Oliveira
10196 et al., 2007).

10197 Exercises

10198 1. The SimpleNLG system produces surface realizations from representations of de-
10199 sired syntactic structure (Gatt and Reiter, 2009). This system can be accessed on
10200 github at <https://github.com/simplenlg/simplenlg>. Download the sys-
10201 tem, and produce realizations of the following examples:

- 10202 (19.12) Call me Ismael.
10203 (19.13) I try all things.
10204 (19.14) I achieve what I can.

10205 Then convert each example to a question. [todo: Can't get SimpleNLG to work with
10206 python anymore]

10207 **Appendix A**

10208 **Probability**

10209 Probability theory provides a way to reason about random events. The sorts of random
10210 events that are typically used to explain probability theory include coin flips, card draws,
10211 and the weather. It may seem odd to think about the choice of a word as akin to the flip of
10212 a coin, particularly if you are the type of person to choose words carefully. But random or
10213 not, language has proven to be extremely difficult to model deterministically. Probability
10214 offers a powerful tool for modeling and manipulating linguistic data.

10215 Probability can be thought of in terms of **random outcomes**: for example, a single coin
10216 flip has two possible outcomes, heads or tails. The set of possible outcomes is the **sample**
10217 **space**, and a subset of the **sample space** is an **event**. For a sequence of two coin flips,
10218 there are four possible outcomes, $\{HH, HT, TH, TT\}$, representing the ordered sequences
10219 heads-head, heads-tails, tails-heads, and tails-tails. The event of getting exactly one head
10220 includes two outcomes: $\{HT, TH\}$.

10221 Formally, a probability is a function from events to the interval between zero and one:
10222 $\Pr : \mathcal{F} \rightarrow [0, 1]$, where \mathcal{F} is the set of possible events. An event that is certain has probabili-
10223 ty one; an event that is impossible has probability zero. For example, the probability of
10224 getting less than three heads on two coin flips is one. Each outcome is also an event (a set
10225 with exactly one element), and for two flips of a fair coin, the probability of each outcome
10226 is,

$$\Pr(\{HH\}) = \Pr(\{HT\}) = \Pr(\{TH\}) = \Pr(\{TT\}) = \frac{1}{4}. \quad [\text{A.1}]$$

10227 **A.1 Probabilities of event combinations**

10228 Because events are sets of outcomes, we can use set-theoretic operations such as com-
10229 plement, intersection, and unions to reason about the probabilities of events and their
10230 combinations.

10231 For any event A , there is a **complement** $\neg A$, such that:

- 10232 • The probability of the union $A \cup \neg A$ is $\Pr(A \cup \neg A) = 1$;
 10233 • The intersection $A \cap \neg A = \emptyset$ is the empty set, and $\Pr(A \cap \neg A) = 0$.

10234 In the coin flip example, the event of obtaining a single head on two flips corresponds to
 10235 the set of outcomes $\{HT, TH\}$; the complement event includes the other two outcomes,
 10236 $\{TT, HH\}$.

10237 **A.1.1 Probabilities of disjoint events**

10238 In general, when two events have an empty intersection, $A \cap B = \emptyset$, they are said to be
 10239 **disjoint**. The probability of the union of two disjoint events is equal to the sum of their
 10240 probabilities,

$$A \cap B = \emptyset \Rightarrow \Pr(A \cup B) = \Pr(A) + \Pr(B). \quad [A.2]$$

10241 This is the **third axiom of probability**, and can be generalized to any countable sequence
 10242 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,

$$\Pr(\{HT, TH\}) = \Pr(\{HT\} \cup \{TH\}) = \Pr(\{HT\}) + \Pr(\{TH\}) \quad [A.3]$$

$$= \frac{1}{4} + \frac{1}{4} = \frac{1}{2}. \quad [A.4]$$

10243 For events that are not disjoint, it is still possible to compute the probability of their
 10244 union:

$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B). \quad [A.5]$$

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 , which we can write 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 $B - (A \cap B)$.

$$\Pr(B) = \Pr(B - (A \cap B)) + \Pr(A \cap B) \quad [A.7]$$

$$\Pr(B - (A \cap B)) = \Pr(B) - \Pr(A \cap B) \quad [A.8]$$

$$[A.9]$$

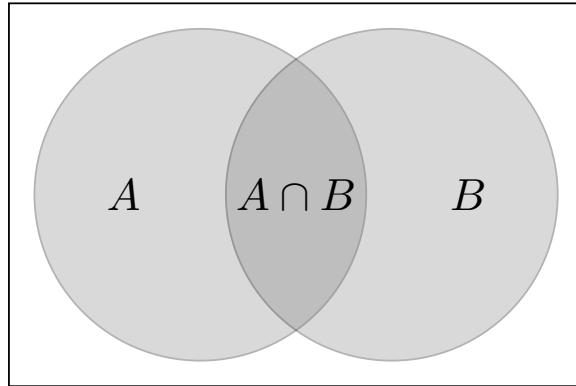


Figure A.1: A visualization of the probability of non-disjoint events A and B .

Substituting this into Equation A.6 gives the desired result:

$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B). \quad [\text{A.10}]$$

10245 A.1.2 Law of total probability

10246 A set of events $\mathcal{B} = \{B_1, B_2, \dots, B_N\}$ is a **partition** of the sample space iff each pair of
 10247 events is disjoint ($B_i \cap B_j = \emptyset$), and the union of the events is the entire sample space.
 10248 The law of total probability states that we can **marginalize** over these events as follows,

$$\Pr(A) = \sum_{B_n \in \mathcal{B}} \Pr(A \cap B_n). \quad [\text{A.11}]$$

10249 Note for any event B , the union $B \cup \neg B$ forms a partition of the sample space. Therefore,
 10250 an important special case of the law of total probability is,

$$\Pr(A) = \Pr(A \cap B) + \Pr(A \cap \neg B). \quad [\text{A.12}]$$

10251 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,

$$\Pr(A \mid B) = \frac{\Pr(A \cap B)}{\Pr(B)}. \quad [\text{A.13}]$$

The **chain rule of probability** states that $\Pr(A \cap B) = \Pr(A | B) \times \Pr(B)$, which is just a rearrangement of terms from Equation A.13. We can apply the chain rule 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 chain rule:

$$\Pr(A | B) = \frac{\Pr(A \cap B)}{\Pr(B)} = \frac{\Pr(B | A) \times \Pr(A)}{\Pr(B)} \quad [\text{A.14}]$$

10252 The terms in Bayes rule have specialized names, which we will occasionally use:

- 10253 • Pr(A) is the **prior**, since it is the probability of event A without knowledge about
10254 whether B happens or not.
- 10255 • Pr($B | A$) is the **likelihood**, the probability of event B given that event A has oc-
10256 curred.
- 10257 • Pr($A | B$) is the **posterior**, since it is the probability of event A with knowledge that
10258 B has occurred.

10259 **Example** Manning and Schütze (1999) provide an example of Bayes' rule in a linguistic
10260 setting. (This same example is usually framed in terms of tests for rare diseases.) Suppose
10261 that you are interested in a rare syntactic construction, such as *parasitic gaps*, which
10262 occur on average once in 100,000 sentences. Here is an example:

10263 (A.1) *Which class did you attend ... without registering for ...?*

10264 Lana Linguist has developed a complicated pattern matcher that attempts to identify
10265 sentences with parasitic gaps. It's pretty good, but it's not perfect:

- 10266 • If a sentence has a parasitic gap, the pattern matcher will find it with probability
10267 0.95. (This is the **recall**, which is one minus the **false positive rate**.)
- 10268 • If the sentence doesn't have a parasitic gap, the pattern matcher will wrongly say it
10269 does with probability 0.005. (This is the **false positive rate**, which is one minus the
10270 **precision**.)

10271 Suppose that Lana's pattern matcher says that a sentence contains a parasitic gap. What
10272 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:

$$\Pr(G | T) = \frac{\Pr(T | G) \times \Pr(G)}{\Pr(T)}. \quad [\text{A.15}]$$

10273 We already know both terms in the numerator: $\Pr(T | G)$ is the recall, which is 0.95; $\Pr(G)$
10274 is the prior, which is 10^{-5} .

10275 We are not given the denominator, but it can be computed using tools developed earlier
10276 in this section. First apply the law of total probability, using the partition $\{G, \neg G\}$:

$$\Pr(T) = \Pr(T \cap G) + \Pr(T \cap \neg G). \quad [\text{A.16}]$$

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:

$$\Pr(T \cap G) = \Pr(T | G) \times \Pr(G) = 0.95 \times 10^{-5} \quad [\text{A.17}]$$

$$\Pr(T \cap \neg G) = \Pr(T | \neg G) \times \Pr(\neg G) = 0.005 \times (1 - 10^{-5}) \approx 0.005 \quad [\text{A.18}]$$

$$\Pr(T) = \Pr(T \cap G) + \Pr(T \cap \neg G) \quad [\text{A.19}]$$

$$= 0.95 \times 10^{-5} + 0.005 \approx 0.005. \quad [\text{A.20}]$$

We now return to Bayes' rule to compute the desired posterior probability,

$$\Pr(G | T) = \frac{\Pr(T | G) \Pr(G)}{\Pr(T)} \quad [\text{A.21}]$$

$$= \frac{0.95 \times 10^{-5}}{0.95 \times 10^{-5} + 0.005 \times (1 - 10^{-5})} \quad [\text{A.22}]$$

$$\approx 0.002. \quad [\text{A.23}]$$

10277 Lana's pattern matcher seems accurate, with false positive and false negative rates
10278 below 5%. Yet the extreme rarity of this phenomenon means that a positive result from
10279 the detector is most likely to be wrong.

10280 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:

$$\Pr(\{HT, HH\}) = \Pr(HT) + \Pr(HH) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2} \quad [A.24]$$

$$\Pr(\{HH, TH\}) = \Pr(HH) + \Pr(TH) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2} \quad [A.25]$$

$$\Pr(\{HT, HH\}) \times \Pr(\{HH, TH\}) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4} \quad [A.26]$$

$$\Pr(\{HT, HH\} \cap \{HH, TH\}) = \Pr(HH) = \frac{1}{4} \quad [A.27]$$

$$= \Pr(\{HT, HH\}) \times \Pr(\{HH, TH\}). \quad [A.28]$$

If $\Pr(A \cap B | C) = \Pr(A | C) \times \Pr(B | C)$, then the events A and B are **conditionally independent**, written $A \perp B | C$. Conditional independence plays a key role in probabilistic models such as Naïve Bayes chapter 2.

A.4 Random variables

Random variables are functions from events to the space \mathbb{R}^n , where \mathbb{R} is the set of real numbers. This subsumes several useful special cases:

- **Indicator random variables** are functions from events to the set $\{0, 1\}$. In the coin flip example, we can define Y as an indicator random variable, for whether 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 $\Pr(X = 1) = \frac{1}{2}$.

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:

$$\int_x p_X(x)dx = 1 \quad [A.29]$$

$$\forall x, p_X(x) \geq 0. \quad [A.30]$$

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:

- We can compute a **marginal probability distribution** $p_A(a) = \sum_b p_{A,B}(a,b)$.
- We can compute a **conditional probability distribution** $p_{A|B}(a | b) = \frac{p_{A,B}(a,b)}{p_B(b)}$.
- 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 sum becomes an integral:

$$E[g(x)] = \int_{\mathcal{X}} g(x)p(x)dx \quad [A.31]$$

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 they use a thermometer with infinite precision, the expected price would be an integral over all possible temperatures,

$$E[\text{price}(x)] = \int_{\mathcal{X}} \min(1, 1+x) \times \text{original-price} \times p(x)dx. \quad [A.32]$$

10316 **A.6 Modeling and estimation**

10317 **Probabilistic models** provide a principled way to reason about random events and ran-
10318 dom variables, and to make predictions about the future. Let's consider the coin toss
10319 example. Each toss can be modeled as a random event, with probability θ of the event H ,
10320 and probability $1 - \theta$ of the complementary event T . If we write a random variable X as
10321 the total number of heads on three coin flips, then the distribution of X depends on θ . In
10322 this case, X is distributed as a **binomial random variable**, meaning that it is drawn from
10323 a binomial distribution, with **parameters** $(\theta, N = 3)$. This is written,

$$X \sim \text{Binomial}(\theta, N = 3). \quad [\text{A.33}]$$

10324 The properties of the binomial distribution enable us to make statements about the X ,
10325 such as its expected value and the likelihood that its value will fall within some interval.

Now suppose that θ is unknown, but we have run an experiment, in which we exe-
 cuted N trials, and obtained x heads. We can **estimate** θ by the principle of **maximum
 likelihood**:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} p_X(x; \theta, N). \quad [\text{A.34}]$$

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,

$$p_X(x; \theta, N) = \frac{N!}{x!(N-x)!} \theta^x (1 - \theta)^{N-x}. \quad [\text{A.35}]$$

10326 This likelihood is proportional to the product of the probability of individual out-
10327 comes: for example, the sequence T, H, H, T, H would have probability $\theta^2(1 - \theta)^3$. The
10328 term $\frac{N!}{x!(N-x)!}$ arises from the many possible orderings by which we could obtain x heads
10329 on N trials. This term does not depend on θ , so it can be ignored during estimation.

In practice, we usually maximize log-likelihood, which is a monotonic function of the
 likelihood. 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.

$$\ell(\theta) = x \log \theta + (N - x) \log(1 - \theta) \quad [\text{A.36}]$$

$$\frac{\partial \ell(\theta)}{\partial \theta} = \frac{x}{\theta} - \frac{N - x}{1 - \theta} \quad [\text{A.37}]$$

$$\frac{N - x}{1 - \theta} = \frac{x}{\theta} \quad [\text{A.38}]$$

$$\frac{N - x}{x} = \frac{1 - \theta}{\theta} \quad [\text{A.39}]$$

$$\frac{N}{x} - 1 = \frac{1}{\theta} - 1 \quad [\text{A.40}]$$

$$\hat{\theta} = \frac{x}{N}. \quad [\text{A.41}]$$

10330 In this case, the maximum likelihood estimate is equal to $\frac{x}{N}$, the fraction of trials that
 10331 came up heads. This intuitive solution is also known as the **relative frequency estimate**,
 10332 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:

$$p(\theta | x; N) = \frac{p(x | \theta) \times p(\theta)}{p(x)} \quad [\text{A.42}]$$

$$\propto p(x | \theta) \times p(\theta) \quad [\text{A.43}]$$

$$\hat{\theta} = \operatorname{argmax}_{\theta} p(x | \theta) \times p(\theta). \quad [\text{A.44}]$$

10333 This is the **maximum a posteriori** (MAP) estimate. Given a form for $p(\theta)$, you can de-
 10334 rive the MAP estimate using the same approach that was used to derive the maximum
 10335 likelihood estimate.

10336 Further reading

10337 A good introduction to probability theory is offered by Manning and Schütze (1999),
 10338 which helped to motivate this section. For more detail, Sharon Goldwater provides an-
 10339 other useful reference, [http://homepages.inf.ed.ac.uk/sgwater/teaching/general/
 10340 probability.pdf](http://homepages.inf.ed.ac.uk/sgwater/teaching/general/probability.pdf). A historical and philosophical perspective on probability is offered
 10341 by Diaconis and Skyrms (2017).

10342 **Appendix B**

10343 **Continuous optimization**

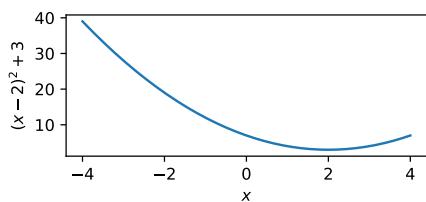
10344 Unconstrained continuous optimization involves solving problems of the form,

$$\min_{\mathbf{x} \in \mathbb{R}^D} f(\mathbf{x}), \quad [\text{B.1}]$$

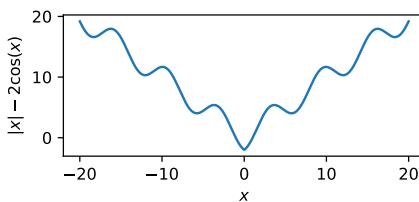
10345 where $\mathbf{x} \in \mathbb{R}^D$ is a vector of D real numbers.

10346 Differentiation is fundamental to continuous optimization. Suppose that at some \mathbf{x}^* ,
10347 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**
10348 **point** of f . For a **convex** function f (defined in § 2.3), $f(\mathbf{x}^*)$ is equal to the global minimum
10349 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



(a) The function $f(x) = (x - 2)^2 + 3$



(b) The function $f(x) = |x| - 2\cos(x)$

Figure B.1: Two functions

function $f(\mathbf{x}) = \frac{1}{2} \|\mathbf{x} - [2, 1]^\top\|^2$. The partials derivatives are,

$$\frac{\partial d}{\partial x_1} = x_1 - 2 \quad [B.2]$$

$$\frac{\partial d}{\partial x_2} = x_2 - 1 \quad [B.3]$$

10350 The unique minimum is $\mathbf{x}^* = [2, 1]^\top$.

10351 For non-convex functions, critical points are not necessarily global minima. A **local**
 10352 **minimum** \mathbf{x}^* is a point at which the function takes a smaller value than at all nearby
 10353 neighbors: formally, \mathbf{x}^* is a local minimum if there is some positive ϵ such that $f(\mathbf{x}^*) \leq$
 10354 $f(\mathbf{x})$ for all \mathbf{x} within distance ϵ of \mathbf{x}^* . Figure B.1b shows the function $f(x) = |x| - 2 \cos(x)$,
 10355 which has many local minima, as well as a unique global minimum at $x = 0$. A critical
 10356 point may also be the local or global maximum of the function; it may be a **saddle point**,
 10357 which is a minimum with respect to at least one coordinate, and a maximum with respect
 10358 to at least one other coordinate; it may be an **inflection point**, which is neither a minimum
 10359 nor maximum. When available, the second derivative of f can help to distinguish these
 10360 cases.

10361 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 $\mathbf{x}^{(t+1)}$ can be computed,

$$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} - \eta^{(t)} \nabla_{\mathbf{x}^{(t)}} f. \quad [B.4]$$

10362 where $\eta^{(t)} > 0$ is a **step size**. If the step size is chosen appropriately, this procedure will
 10363 find the global minimum of a differentiable convex function. For non-convex functions,
 10364 gradient descent will find a local minimum. The extension to non-differentiable convex
 10365 functions is discussed in § 2.3.

10366 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 is sometimes 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 was 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 perceptron, but the goal at each step is to make the most conservative update that gives zero margin loss on the current example.¹ This 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 $\boldsymbol{\theta}^{(i-1)}$, the optimal solution is simply to set $\boldsymbol{\theta}^* = \boldsymbol{\theta}^{(i-1)}$. Let us focus on the case where $\boldsymbol{\theta}^{(i-1)}$ gives non-zero margin loss on example i . The Lagrangian for this problem is,

$$L(\boldsymbol{\theta}, \lambda) = \frac{1}{2} \|\boldsymbol{\theta} - \boldsymbol{\theta}^{(i-1)}\|^2 + \lambda \ell^{(i)}(\boldsymbol{\theta}), \quad [\text{B.14}]$$

Holding λ constant, we can solve for $\boldsymbol{\theta}$ by differentiating,

$$\nabla_{\boldsymbol{\theta}} L = \boldsymbol{\theta} - \boldsymbol{\theta}^{(i-1)} + \lambda \frac{\partial}{\partial \boldsymbol{\theta}} \ell^{(i)}(\boldsymbol{\theta}) \quad [\text{B.15}]$$

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}^{(i-1)} + \lambda \boldsymbol{\delta}, \quad [\text{B.16}]$$

where $\boldsymbol{\delta} = \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - \mathbf{f}(\mathbf{x}^{(i)}, \hat{y})$ and $\hat{y} = \operatorname{argmax}_{y \neq y^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y)$.

The Lagrange multiplier λ acts as the learning rate in a perceptron-like update to $\boldsymbol{\theta}$. We can compute it by plugging $\boldsymbol{\theta}^*$ back into the Lagrangian, obtaining the dual function,

$$D(\lambda) = \frac{1}{2} \|\boldsymbol{\theta}^{(i-1)} + \lambda \boldsymbol{\delta} - \boldsymbol{\theta}^{(i-1)}\|^2 + \lambda(1 - (\boldsymbol{\theta}^{(i-1)} + \lambda \boldsymbol{\delta}) \cdot \boldsymbol{\delta}) \quad [\text{B.17}]$$

$$= \frac{\lambda^2}{2} \|\boldsymbol{\delta}\|^2 - \lambda^2 \|\boldsymbol{\delta}\|^2 + \lambda(1 - \boldsymbol{\theta}^{(i-1)} \cdot \boldsymbol{\delta}) \quad [\text{B.18}]$$

$$= -\frac{\lambda^2}{2} \|\boldsymbol{\delta}\|^2 + \lambda \ell^{(i)}(\boldsymbol{\theta}^{(i-1)}). \quad [\text{B.19}]$$

Differentiating and solving for λ ,

$$\frac{\partial D}{\partial \lambda} = -\lambda \|\boldsymbol{\delta}\|^2 + \ell^{(i)}(\boldsymbol{\theta}^{(i-1)}) \quad [\text{B.20}]$$

$$\lambda^* = \frac{\ell^{(i)}(\boldsymbol{\theta}^{(i-1)})}{\|\boldsymbol{\delta}\|^2}. \quad [\text{B.21}]$$

The complete update equation for the learning algorithm is therefore:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}^{(i-1)} + \frac{\ell^{(i)}(\boldsymbol{\theta}^{(i-1)})}{\|\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - \mathbf{f}(\mathbf{x}^{(i)}, \hat{y})\|^2} (\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - \mathbf{f}(\mathbf{x}^{(i)}, \hat{y})). \quad [\text{B.22}]$$

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.

10384

Bibliography

- 10385 Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis,
10386 J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia,
10387 R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore,
10388 D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A.
10389 Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Watten-
10390 berg, M. Wicke, Y. Yu, and X. Zheng (2016). Tensorflow: Large-scale machine learning
10391 on heterogeneous distributed systems. *CoRR abs/1603.04467*.
- 10392 Abend, O. and A. Rappoport (2017). The state of the art in semantic representation. In
10393 *Proceedings of the Association for Computational Linguistics (ACL)*.
- 10394 Abney, S., R. E. Schapire, and Y. Singer (1999). Boosting applied to tagging and PP attach-
10395 ment. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.
10396 132–134.
- 10397 Abney, S. P. (1987). *The English noun phrase in its sentential aspect*. Ph. D. thesis, Mas-
10398 sachusetts Institute of Technology.
- 10399 Abney, S. P. and M. Johnson (1991). Memory requirements and local ambiguities of pars-
10400 ing strategies. *Journal of Psycholinguistic Research* 20(3), 233–250.
- 10401 Adafre, S. F. and M. De Rijke (2006). Finding similar sentences across multiple languages
10402 in wikipedia. In *Proceedings of the Workshop on NEW TEXT Wikis and blogs and other*
10403 *dynamic text sources*.
- 10404 Ahn, D. (2006). The stages of event extraction. In *Proceedings of the Workshop on Annotating*
10405 *and Reasoning about Time and Events*, pp. 1–8. Association for Computational Linguistics.
- 10406 Aikhenvald, A. Y. (2004). *Evidentiality*. Oxford University Press.
- 10407 Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on*
10408 *Automatic Control* 19(6), 716–723.

- 10409 Akmajian, A., R. A. Demers, A. K. Farmer, and R. M. Harnish (2010). *Linguistics: An
10410 introduction to language and communication* (Sixth ed.). Cambridge, MA: MIT press.
- 10411 Allauzen, C., M. Riley, J. Schalkwyk, W. Skut, and M. Mohri (2007). OpenFst: A gen-
10412 eral and efficient weighted finite-state transducer library. In *International Conference on
10413 Implementation and Application of Automata*, pp. 11–23. Springer.
- 10414 Allen, J. F. (1984). Towards a general theory of action and time. *Artificial intelligence* 23(2),
10415 123–154.
- 10416 Allen, J. F., B. W. Miller, E. K. Ringger, and T. Sikorski (1996). A robust system for natural
10417 spoken dialogue. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
10418 62–70.
- 10419 Allen, J. F., L. K. Schubert, G. Ferguson, P. Heeman, C. H. Hwang, T. Kato, M. Light,
10420 N. Martin, B. Miller, M. Poesio, and D. Traum (1995). The TRAINS project: A case
10421 study in building a conversational planning agent. *Journal of Experimental & Theoretical
10422 Artificial Intelligence* 7(1), 7–48.
- 10423 Alm, C. O., D. Roth, and R. Sproat (2005). Emotions from text: machine learning for
10424 text-based emotion prediction. In *Proceedings of Empirical Methods for Natural Language
10425 Processing (EMNLP)*, pp. 579–586.
- 10426 Aluísio, S., J. Pelizzoni, A. Marchi, L. de Oliveira, R. Manenti, and V. Marquiafável (2003).
10427 An account of the challenge of tagging a reference corpus for Brazilian Portuguese.
10428 *Computational Processing of the Portuguese Language*, 194–194.
- 10429 Anand, P., M. Walker, R. Abbott, J. E. Fox Tree, R. Bowman, and M. Minor (2011). Cats rule
10430 and dogs drool!: Classifying stance in online debate. In *Proceedings of the 2nd Workshop
10431 on Computational Approaches to Subjectivity and Sentiment Analysis*, Portland, Oregon, pp.
10432 1–9. Association for Computational Linguistics.
- 10433 Anandkumar, A. and R. Ge (2016). Efficient approaches for escaping higher order saddle
10434 points in non-convex optimization. In *Proceedings of the Conference On Learning Theory
10435 (COLT)*, pp. 81–102.
- 10436 Anandkumar, A., R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky (2014). Tensor decompo-
10437 sitions for learning latent variable models. *The Journal of Machine Learning Research* 15(1),
10438 2773–2832.
- 10439 Ando, R. K. and T. Zhang (2005). A framework for learning predictive structures from
10440 multiple tasks and unlabeled data. *The Journal of Machine Learning Research* 6, 1817–
10441 1853.

- 10442 Andor, D., C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and
10443 M. Collins (2016). Globally normalized transition-based neural networks. In *Proceedings*
10444 of the Association for Computational Linguistics (ACL), pp. 2442–2452.
- 10445 Angeli, G., P. Liang, and D. Klein (2010). A simple domain-independent probabilistic ap-
10446 proach to generation. In *Proceedings of Empirical Methods for Natural Language Processing*
10447 (EMNLP), pp. 502–512.
- 10448 Antol, S., A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh
10449 (2015). Vqa: Visual question answering. In *Proceedings of the International Conference on*
10450 *Computer Vision (ICCV)*, pp. 2425–2433.
- 10451 Aronoff, M. (1976). *Word formation in generative grammar*. MIT Press.
- 10452 Arora, S. and B. Barak (2009). *Computational complexity: a modern approach*. Cambridge
10453 University Press.
- 10454 Arora, S., R. Ge, Y. Halpern, D. Mimno, A. Moitra, D. Sontag, Y. Wu, and M. Zhu (2013).
10455 A practical algorithm for topic modeling with provable guarantees. In *Proceedings of the*
10456 *International Conference on Machine Learning (ICML)*, pp. 280–288.
- 10457 Arora, S., Y. Li, Y. Liang, T. Ma, and A. Risteski (2016). Linear algebraic structure of word
10458 senses, with applications to polysemy. *arXiv preprint arXiv:1601.03764*.
- 10459 Artstein, R. and M. Poesio (2008). Inter-coder agreement for computational linguistics.
10460 *Computational Linguistics* 34(4), 555–596.
- 10461 Artzi, Y. and L. Zettlemoyer (2013). Weakly supervised learning of semantic parsers for
10462 mapping instructions to actions. *Transactions of the Association for Computational Linguis-*
10463 *tics* 1, 49–62.
- 10464 Attardi, G. (2006). Experiments with a multilanguage non-projective dependency parser.
10465 In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 166–170.
- 10466 Auer, P. (2013). *Code-switching in conversation: Language, interaction and identity*. Routledge.
- 10467 Auer, S., C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives (2007). Dbpedia: A
10468 nucleus for a web of open data. *The semantic web*, 722–735.
- 10469 Austin, J. L. (1962). *How to do things with words*. Oxford University Press.
- 10470 Aw, A., M. Zhang, J. Xiao, and J. Su (2006). A phrase-based statistical model for SMS text
10471 normalization. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
10472 33–40.

- 10473 Ba, J. L., J. R. Kiros, and G. E. Hinton (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.
- 10475 Bagga, A. and B. Baldwin (1998a). Algorithms for scoring coreference chains. In *Proceedings of the Language Resources and Evaluation Conference*, pp. 563–566.
- 10477 Bagga, A. and B. Baldwin (1998b). Entity-based cross-document coreferencing using the vector space model. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp. 79–85.
- 10480 Bahdanau, D., K. Cho, and Y. Bengio (2014). Neural machine translation by jointly learning to align and translate. In *Neural Information Processing Systems (NIPS)*.
- 10482 Baldwin, T. and S. N. Kim (2010). Multiword expressions. In *Handbook of natural language processing*, Volume 2, pp. 267–292. Boca Raton, USA: CRC Press.
- 10484 Balle, B., A. Quattoni, and X. Carreras (2011). A spectral learning algorithm for finite state transducers. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML)*, pp. 156–171.
- 10487 Banarescu, L., C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider (2013, August). Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, Sofia, Bulgaria, pp. 178–186. Association for Computational Linguistics.
- 10492 Banko, M., M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni (2007). Open information extraction from the web. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2670–2676.
- 10495 Bansal, N., A. Blum, and S. Chawla (2004). Correlation clustering. *Machine Learning* 56(1–3), 89–113.
- 10497 Barber, D. (2012). *Bayesian reasoning and machine learning*. Cambridge University Press.
- 10498 Barman, U., A. Das, J. Wagner, and J. Foster (2014, October). Code mixing: A challenge for language identification in the language of social media. In *Proceedings of the First Workshop on Computational Approaches to Code Switching*, Doha, Qatar, pp. 13–23. Association for Computational Linguistics.
- 10502 Barnickel, T., J. Weston, R. Collobert, H.-W. Mewes, and V. Stümpflen (2009). Large scale application of neural network based semantic role labeling for automated relation extraction from biomedical texts. *PLoS One* 4(7), e6393.

- 10505 Baron, A. and P. Rayson (2008). Vard2: A tool for dealing with spelling variation in his-
10506 torical corpora. In *Postgraduate conference in corpus linguistics*.
- 10507 Baroni, M., R. Bernardi, and R. Zamparelli (2014). Frege in space: A program for compo-
10508 sitional distributional semantics. *Linguistic Issues in Language Technologies*.
- 10509 Baroni, M. and R. Zamparelli (2010). Nouns are vectors, adjectives are matrices: Rep-
10510 resenting adjective-noun constructions in semantic space. In *Proceedings of Empirical
10511 Methods for Natural Language Processing (EMNLP)*, pp. 1183–1193.
- 10512 Barzilay, R. and M. Lapata (2008, mar). Modeling local coherence: An Entity-Based ap-
10513 proach. *Computational Linguistics* 34(1), 1–34.
- 10514 Barzilay, R. and K. R. McKeown (2005). Sentence fusion for multidocument news summa-
10515 rization. *Computational Linguistics* 31(3), 297–328.
- 10516 Beesley, K. R. and L. Karttunen (2003). *Finite-state morphology*. Stanford, CA: Center for
10517 the Study of Language and Information.
- 10518 Bejan, C. A. and S. Harabagiu (2014). Unsupervised event coreference resolution. *Compu-
10519 tational Linguistics* 40(2), 311–347.
- 10520 Bell, E. T. (1934). Exponential numbers. *The American Mathematical Monthly* 41(7), 411–419.
- 10521 Bender, E. M. (2013, jun). *Linguistic Fundamentals for Natural Language Processing: 100
10522 Essentials from Morphology and Syntax*, Volume 6 of *Synthesis Lectures on Human Language
10523 Technologies*. Morgan & Claypool Publishers.
- 10524 Bengio, S., O. Vinyals, N. Jaitly, and N. Shazeer (2015). Scheduled sampling for sequence
10525 prediction with recurrent neural networks. In *Neural Information Processing Systems
10526 (NIPS)*, pp. 1171–1179.
- 10527 Bengio, Y., R. Ducharme, P. Vincent, and C. Janvin (2003). A neural probabilistic language
10528 model. *The Journal of Machine Learning Research* 3, 1137–1155.
- 10529 Bengio, Y., P. Simard, and P. Frasconi (1994). Learning long-term dependencies with gra-
10530 dient descent is difficult. *IEEE Transactions on Neural Networks* 5(2), 157–166.
- 10531 Bengtsson, E. and D. Roth (2008). Understanding the value of features for coreference
10532 resolution. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
10533 pp. 294–303.
- 10534 Benjamini, Y. and Y. Hochberg (1995). Controlling the false discovery rate: a practical and
10535 powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B
10536 (Methodological)*, 289–300.

- 10537 Berant, J., A. Chou, R. Frostig, and P. Liang (2013). Semantic parsing on freebase from
 10538 question-answer pairs. In *Proceedings of Empirical Methods for Natural Language Processing*
 10539 (*EMNLP*), pp. 1533–1544.
- 10540 Berant, J., V. Srikumar, P.-C. Chen, A. Vander Linden, B. Harding, B. Huang, P. Clark, and
 10541 C. D. Manning (2014). Modeling biological processes for reading comprehension. In
 10542 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 10543 Berg-Kirkpatrick, T., A. Bouchard-Côté, J. DeNero, and D. Klein (2010). Painless unsuper-
 10544 vised learning with features. In *Proceedings of the North American Chapter of the Associa-*
 10545 *tion for Computational Linguistics (NAACL)*, pp. 582–590.
- 10546 Berg-Kirkpatrick, T., D. Burkett, and D. Klein (2012). An empirical investigation of sta-
 10547 tistical significance in NLP. In *Proceedings of Empirical Methods for Natural Language*
 10548 *Processing (EMNLP)*, pp. 995–1005.
- 10549 Berger, A. L., V. J. D. Pietra, and S. A. D. Pietra (1996). A maximum entropy approach to
 10550 natural language processing. *Computational linguistics* 22(1), 39–71.
- 10551 Bergsma, S., D. Lin, and R. Goebel (2008). Distributional identification of non-referential
 10552 pronouns. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 10–18.
- 10553 Bernardi, R., R. Cakici, D. Elliott, A. Erdem, E. Erdem, N. Ikizler-Cinbis, F. Keller, A. Mus-
 10554 cat, and B. Plank (2016). Automatic description generation from images: A survey of
 10555 models, datasets, and evaluation measures. *Journal of Artificial Intelligence Research* 55,
 10556 409–442.
- 10557 Bertsekas, D. P. (2012). Incremental gradient, subgradient, and proximal methods for
 10558 convex optimization: A survey. See Sra et al. (2012).
- 10559 Bhatia, P., R. Guthrie, and J. Eisenstein (2016). Morphological priors for probabilistic neu-
 10560 ral word embeddings. In *Proceedings of Empirical Methods for Natural Language Processing*
 10561 (*EMNLP*).
- 10562 Bhatia, P., Y. Ji, and J. Eisenstein (2015). Better document-level sentiment analysis from
 10563 first discourse parsing. In *Proceedings of Empirical Methods for Natural Language Processing*
 10564 (*EMNLP*).
- 10565 Biber, D. (1991). *Variation across speech and writing*. Cambridge University Press.
- 10566 Bird, S., E. Klein, and E. Loper (2009). *Natural language processing with Python*. California:
 10567 O'Reilly Media.
- 10568 Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

- 10569 Björkelund, A. and P. Nugues (2011). Exploring lexicalized features for coreference reso-
10570 lution. In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 45–50.
- 10571 Blackburn, P. and J. Bos (2005). *Representation and inference for natural language: A first*
10572 *course in computational semantics*. CSLI.
- 10573 Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM* 55(4), 77–84.
- 10574 Blei, D. M. (2014). Build, compute, critique, repeat: Data analysis with latent variable
10575 models. *Annual Review of Statistics and Its Application* 1, 203–232.
- 10576 Blei, D. M., A. Y. Ng, and M. I. Jordan (2003). Latent dirichlet allocation. *the Journal of*
10577 *machine Learning research* 3, 993–1022.
- 10578 Blitzer, J., M. Dredze, and F. Pereira (2007). Biographies, bollywood, boom-boxes and
10579 blenders: Domain adaptation for sentiment classification. In *Proceedings of the Associa-*
10580 *tion for Computational Linguistics (ACL)*, pp. 440–447.
- 10581 Blum, A. and T. Mitchell (1998). Combining labeled and unlabeled data with co-training.
10582 In *Proceedings of the Conference On Learning Theory (COLT)*, pp. 92–100.
- 10583 Bobrow, D. G., R. M. Kaplan, M. Kay, D. A. Norman, H. Thompson, and T. Winograd
10584 (1977). Gus, a frame-driven dialog system. *Artificial intelligence* 8(2), 155–173.
- 10585 Böhmová, A., J. Hajič, E. Hajičová, and B. Hladká (2003). The Prague dependency tree-
10586 bank. In *Treebanks, Text, Speech and Language Technology*, pp. 103–127. Springer.
- 10587 Bochnet, B. (2010). Very high accuracy and fast dependency parsing is not a contradiction.
10588 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.
10589 89–97.
- 10590 Boitet, C. (1988). Pros and cons of the pivot and transfer approaches in multilingual ma-
10591 chine translation. *Readings in machine translation*, 273–279.
- 10592 Bojanowski, P., E. Grave, A. Joulin, and T. Mikolov (2017). Enriching word vectors with
10593 subword information. *Transactions of the Association for Computational Linguistics* 5, 135–
10594 146.
- 10595 Bollacker, K., C. Evans, P. Paritosh, T. Sturge, and J. Taylor (2008). Freebase: a collabora-
10596 tively created graph database for structuring human knowledge. In *Proceedings of the*
10597 *ACM International Conference on Management of Data (SIGMOD)*, pp. 1247–1250. AcM.
- 10598 Bolukbasi, T., K.-W. Chang, J. Y. Zou, V. Saligrama, and A. T. Kalai (2016). Man is to
10599 computer programmer as woman is to homemaker? debiasing word embeddings. In
10600 *Neural Information Processing Systems (NIPS)*, pp. 4349–4357.

- 10601 Bordes, A., N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko (2013). Translating
 10602 embeddings for modeling multi-relational data. In *Neural Information Processing Systems*
 10603 (*NIPS*), pp. 2787–2795.
- 10604 Bordes, A., J. Weston, R. Collobert, Y. Bengio, et al. (2011). Learning structured embed-
 10605 dings of knowledge bases. In *Proceedings of the National Conference on Artificial Intelligence*
 10606 (*AAAI*), pp. 301–306.
- 10607 Borges, J. L. (1993). *Other Inquisitions 1937–1952*. University of Texas Press. Translated by
 10608 Ruth L. C. Simms.
- 10609 Botha, J. A. and P. Blunsom (2014). Compositional morphology for word representations
 10610 and language modelling. In *Proceedings of the International Conference on Machine Learn-*
 10611 *ing (ICML)*.
- 10612 Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*,
 10613 pp. 421–436. Springer.
- 10614 Bottou, L., F. E. Curtis, and J. Nocedal (2016). Optimization methods for large-scale ma-
 10615 chine learning. *arXiv preprint arXiv:1606.04838*.
- 10616 Bowman, S. R., L. Vilnis, O. Vinyals, A. Dai, R. Jozefowicz, and S. Bengio (2016). Gen-
 10617 erating sentences from a continuous space. In *Proceedings of the Conference on Natural*
 10618 *Language Learning (CoNLL)*, pp. 10–21.
- 10619 boyd, d. and K. Crawford (2012). Critical questions for big data. *Information, Communica-*
 10620 *tion & Society* 15(5), 662–679.
- 10621 Boyd, S. and L. Vandenberghe (2004). *Convex Optimization*. New York: Cambridge Uni-
 10622 *versity Press*.
- 10623 Branavan, S., H. Chen, J. Eisenstein, and R. Barzilay (2009). Learning document-level
 10624 semantic properties from free-text annotations. *Journal of Artificial Intelligence Re-*
 10625 *search* 34(2), 569–603.
- 10626 Branavan, S. R., H. Chen, L. S. Zettlemoyer, and R. Barzilay (2009). Reinforcement learning
 10627 for mapping instructions to actions. In *Proceedings of the Association for Computational*
 10628 *Linguistics (ACL)*, pp. 82–90.
- 10629 Braud, C., O. Lacroix, and A. Søgaard (2017). Does syntax help discourse segmenta-
 10630 tion? not so much. In *Proceedings of Empirical Methods for Natural Language Processing*
 10631 (*EMNLP*), pp. 2432–2442.
- 10632 Brown, P. F., J. Cocke, S. A. D. Pietra, V. J. D. Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer,
 10633 and P. S. Roossin (1990). A statistical approach to machine translation. *Computational*
 10634 *linguistics* 16(2), 79–85.

- 10635 Brown, P. F., P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai (1992). Class-based
10636 n-gram models of natural language. *Computational linguistics* 18(4), 467–479.
- 10637 Brown, P. F., V. J. D. Pietra, S. A. D. Pietra, and R. L. Mercer (1993). The mathematics
10638 of statistical machine translation: Parameter estimation. *Computational linguistics* 19(2),
10639 263–311.
- 10640 Brun, C. and C. Roux (2014). Décomposition des “hash tags” pour l’amélioration de la
10641 classification en polarité des “tweets”. *Proceedings of Traitement Automatique des Langues
10642 Naturelles*, 473–478.
- 10643 Bruni, E., N.-K. Tran, and M. Baroni (2014). Multimodal distributional semantics. *Journal
10644 of Artificial Intelligence Research* 49(2014), 1–47.
- 10645 Bullinaria, J. A. and J. P. Levy (2007). Extracting semantic representations from word co-
10646 occurrence statistics: A computational study. *Behavior research methods* 39(3), 510–526.
- 10647 Bunescu, R. C. and R. J. Mooney (2005). A shortest path dependency kernel for relation
10648 extraction. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
10649 pp. 724–731.
- 10650 Bunescu, R. C. and M. Pasca (2006). Using encyclopedic knowledge for named entity
10651 disambiguation. In *Proceedings of the European Chapter of the Association for Computational
10652 Linguistics (EACL)*, pp. 9–16.
- 10653 Burstein, J., D. Marcu, and K. Knight (2003). Finding the WRITE stuff: Automatic identi-
10654 fication of discourse structure in student essays. *IEEE Intelligent Systems* 18(1), 32–39.
- 10655 Burstein, J., J. Tetreault, and S. Andreyev (2010). Using entity-based features to model
10656 coherence in student essays. In *Human language technologies: The 2010 annual conference
10657 of the North American chapter of the Association for Computational Linguistics*, pp. 681–684.
10658 Association for Computational Linguistics.
- 10659 Burstein, J., J. Tetreault, and M. Chodorow (2013). Holistic discourse coherence annotation
10660 for noisy essay writing. *Dialogue & Discourse* 4(2), 34–52.
- 10661 Cai, Q. and A. Yates (2013). Large-scale semantic parsing via schema matching and lexicon
10662 extension. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 423–
10663 433.
- 10664 Caliskan, A., J. J. Bryson, and A. Narayanan (2017). Semantics derived automatically from
10665 language corpora contain human-like biases. *Science* 356(6334), 183–186.
- 10666 Canny, J. (1987). A computational approach to edge detection. In *Readings in Computer
10667 Vision*, pp. 184–203. Elsevier.

- 10668 Cappé, O. and E. Moulines (2009). On-line expectation–maximization algorithm for latent
10669 data models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 71(3),
10670 593–613.
- 10671 Carbonell, J. and J. Goldstein (1998). The use of mmr, diversity-based reranking for re-
10672 ordering documents and producing summaries. In *Proceedings of ACM SIGIR conference*
10673 on Research and development in information retrieval, pp. 335–336.
- 10674 Carbonell, J. R. (1970). Mixed-initiative man-computer instructional dialogues. Technical
10675 report, BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS.
- 10676 Cardie, C. and K. Wagstaff (1999). Noun phrase coreference as clustering. In *Proceedings*
10677 of *Empirical Methods for Natural Language Processing (EMNLP)*, pp. 82–89.
- 10678 Carletta, J. (1996). Assessing agreement on classification tasks: the kappa statistic. *Com-
10679 putational linguistics* 22(2), 249–254.
- 10680 Carletta, J. (2007). Unleashing the killer corpus: experiences in creating the multi-
10681 everything ami meeting corpus. *Language Resources and Evaluation* 41(2), 181–190.
- 10682 Carlson, L. and D. Marcu (2001). Discourse tagging reference manual. Technical Report
10683 ISI-TR-545, Information Sciences Institute.
- 10684 Carlson, L., M. E. Okurowski, and D. Marcu (2002). RST discourse treebank. Linguistic
10685 Data Consortium, University of Pennsylvania.
- 10686 Carpenter, B. (1997). *Type-logical semantics*. Cambridge, MA: MIT Press.
- 10687 Carreras, X., M. Collins, and T. Koo (2008). Tag, dynamic programming, and the percep-
10688 tron for efficient, feature-rich parsing. In *Proceedings of the Conference on Natural Language*
10689 *Learning (CoNLL)*, pp. 9–16.
- 10690 Carreras, X. and L. Màrquez (2005). Introduction to the conll-2005 shared task: Semantic
10691 role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language*
10692 *Learning*, pp. 152–164. Association for Computational Linguistics.
- 10693 Carroll, L. (1917). *Through the looking glass: And what Alice found there*. Chicago: Rand,
10694 McNally.
- 10695 Chambers, N. and D. Jurafsky (2008). Jointly combining implicit constraints improves
10696 temporal ordering. In *Proceedings of Empirical Methods for Natural Language Processing*
10697 (*EMNLP*), pp. 698–706.
- 10698 Chang, K.-W., A. Krishnamurthy, A. Agarwal, H. Daume III, and J. Langford (2015).
10699 Learning to search better than your teacher. In *Proceedings of the International Confer-
10700 ence on Machine Learning (ICML)*.

- 10701 Chang, M.-W., L. Ratinov, and D. Roth (2007). Guiding semi-supervision with constraint-
10702 driven learning. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
10703 280–287.
- 10704 Chang, M.-W., L.-A. Ratinov, N. Rizzolo, and D. Roth (2008). Learning and inference with
10705 constraints. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp.
10706 1513–1518.
- 10707 Chapman, W. W., W. Bridewell, P. Hanbury, G. F. Cooper, and B. G. Buchanan (2001). A
10708 simple algorithm for identifying negated findings and diseases in discharge summaries.
10709 *Journal of biomedical informatics* 34(5), 301–310.
- 10710 Charniak, E. (1997). Statistical techniques for natural language parsing. *AI magazine* 18(4),
10711 33–43.
- 10712 Charniak, E. and M. Johnson (2005). Coarse-to-fine n-best parsing and maxent discrimi-
10713 native reranking. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
10714 173–180.
- 10715 Chelba, C. and A. Acero (2006). Adaptation of maximum entropy capitalizer: Little data
10716 can help a lot. *Computer Speech & Language* 20(4), 382–399.
- 10717 Chelba, C., T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson (2013).
10718 One billion word benchmark for measuring progress in statistical language modeling.
10719 *arXiv preprint arXiv:1312.3005*.
- 10720 Chen, D., J. Bolton, and C. D. Manning (2016). A thorough examination of the CNN/Daily
10721 Mail reading comprehension task. In *Proceedings of the Association for Computational
10722 Linguistics (ACL)*.
- 10723 Chen, D. and C. D. Manning (2014). A fast and accurate dependency parser using neural
10724 networks. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
10725 pp. 740–750.
- 10726 Chen, D. L. and R. J. Mooney (2008). Learning to sportscast: a test of grounded language
10727 acquisition. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp.
10728 128–135.
- 10729 Chen, H., S. Branavan, R. Barzilay, and D. R. Karger (2009). Content modeling using latent
10730 permutations. *Journal of Artificial Intelligence Research* 36(1), 129–163.
- 10731 Chen, M., Z. Xu, K. Weinberger, and F. Sha (2012). Marginalized denoising autoencoders
10732 for domain adaptation. In *Proceedings of the International Conference on Machine Learning
10733 (ICML)*.

- 10734 Chen, M. X., O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones, N. Parmar,
 10735 M. Schuster, Z. Chen, Y. Wu, and M. Hughes (2018). The best of both worlds: Combin-
 10736 ing recent advances in neural machine translation. In *Proceedings of the Association for*
 10737 *Computational Linguistics (ACL)*.
- 10738 Chen, S. F. and J. Goodman (1999). An empirical study of smoothing techniques for lan-
 10739 guage modeling. *Computer Speech & Language* 13(4), 359–393.
- 10740 Chen, T. and C. Guestrin (2016). Xgboost: A scalable tree boosting system. In *Proceedings*
 10741 *of Knowledge Discovery and Data Mining (KDD)*, pp. 785–794.
- 10742 Chen, X., X. Qiu, C. Zhu, P. Liu, and X. Huang (2015). Long short-term memory neural
 10743 networks for chinese word segmentation. In *Proceedings of Empirical Methods for Natural*
 10744 *Language Processing (EMNLP)*, pp. 1197–1206.
- 10745 Chen, Y., S. Gilroy, A. Malletti, K. Knight, and J. May (2018). Recurrent neural networks
 10746 as weighted language recognizers. In *Proceedings of the North American Chapter of the*
 10747 *Association for Computational Linguistics (NAACL)*.
- 10748 Chen, Z. and H. Ji (2009). Graph-based event coreference resolution. In *Proceedings of*
 10749 *the 2009 Workshop on Graph-based Methods for Natural Language Processing*, pp. 54–57.
 10750 Association for Computational Linguistics.
- 10751 Cheng, X. and D. Roth (2013). Relational inference for wikification. In *Proceedings of*
 10752 *Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1787–1796.
- 10753 Chiang, D. (2007). Hierarchical phrase-based translation. *Computational Linguistics* 33(2),
 10754 201–228.
- 10755 Chiang, D., J. Graehl, K. Knight, A. Pauls, and S. Ravi (2010). Bayesian inference for
 10756 finite-state transducers. In *Proceedings of the North American Chapter of the Association for*
 10757 *Computational Linguistics (NAACL)*, pp. 447–455.
- 10758 Cho, K. (2015). Natural language understanding with distributed representation.
 10759 *CoRR abs/1511.07916*.
- 10760 Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and
 10761 Y. Bengio (2014). Learning phrase representations using rnn encoder-decoder for sta-
 10762 tistical machine translation. In *Proceedings of Empirical Methods for Natural Language*
 10763 *Processing (EMNLP)*.
- 10764 Chomsky, N. (1957). *Syntactic structures*. The Hague: Mouton & Co.
- 10765 Chomsky, N. (1982). *Some concepts and consequences of the theory of government and binding*,
 10766 Volume 6. MIT press.

- 10767 Choromanska, A., M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun (2015). The loss
10768 surfaces of multilayer networks. In *Proceedings of Artificial Intelligence and Statistics (AIS-*
10769 *TATS)*, pp. 192–204.
- 10770 Christensen, J., S. Soderland, O. Etzioni, et al. (2010). Semantic role labeling for open
10771 information extraction. In *Proceedings of the Workshop on Formalisms and Methodology for*
10772 *Learning by Reading*, pp. 52–60. Association for Computational Linguistics.
- 10773 Chu, Y.-J. and T.-H. Liu (1965). On shortest arborescence of a directed graph. *Scientia*
10774 *Sinica* 14(10), 1396–1400.
- 10775 Chung, C. and J. W. Pennebaker (2007). The psychological functions of function words.
10776 In K. Fiedler (Ed.), *Social communication*, pp. 343–359. New York and Hove: Psychology
10777 Press.
- 10778 Church, K. (2011). A pendulum swung too far. *Linguistic Issues in Language Technology* 6(5),
10779 1–27.
- 10780 Church, K. W. (2000). Empirical estimates of adaptation: the chance of two Noriegas
10781 is closer to $p/2$ than p^2 . In *Proceedings of the International Conference on Computational*
10782 *Linguistics (COLING)*, pp. 180–186.
- 10783 Church, K. W. and P. Hanks (1990). Word association norms, mutual information, and
10784 lexicography. *Computational linguistics* 16(1), 22–29.
- 10785 Ciaramita, M. and M. Johnson (2003). Supersense tagging of unknown nouns in wordnet.
10786 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 168–
10787 175.
- 10788 Clark, K. and C. D. Manning (2015). Entity-centric coreference resolution with model
10789 stacking. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1405–
10790 1415.
- 10791 Clark, K. and C. D. Manning (2016). Improving coreference resolution by learning entity-
10792 level distributed representations. In *Proceedings of the Association for Computational Lin-*
10793 *guistics (ACL)*.
- 10794 Clark, P. (2015). Elementary school science and math tests as a driver for ai: take the aristo
10795 challenge! In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp.
10796 4019–4021.
- 10797 Clarke, J., D. Goldwasser, M.-W. Chang, and D. Roth (2010). Driving semantic parsing
10798 from the world’s response. In *Proceedings of the Conference on Natural Language Learning*
10799 (*CoNLL*), pp. 18–27.

- 10800 Clarke, J. and M. Lapata (2008). Global inference for sentence compression: An integer
10801 linear programming approach. *Journal of Artificial Intelligence Research* 31, 399–429.
- 10802 Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and psychologi-*
10803 *cal measurement* 20(1), 37–46.
- 10804 Cohen, S. (2016). *Bayesian analysis in natural language processing*. Synthesis Lectures on
10805 Human Language Technologies. San Rafael, CA: Morgan & Claypool Publishers.
- 10806 Cohen, S. B., C. Gómez-Rodríguez, and G. Satta (2012). Elimination of spurious ambiguity
10807 in transition-based dependency parsing. *CoRR abs/1206.6735*.
- 10808 Collier, N., C. Nobata, and J.-i. Tsujii (2000). Extracting the names of genes and gene
10809 products with a hidden markov model. In *Proceedings of the International Conference on*
10810 *Computational Linguistics (COLING)*, pp. 201–207.
- 10811 Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *Proceed-*
10812 *ings of the Association for Computational Linguistics (ACL)*, pp. 16–23.
- 10813 Collins, M. (2002). Discriminative training methods for hidden markov models: theory
10814 and experiments with perceptron algorithms. In *Proceedings of Empirical Methods for*
10815 *Natural Language Processing (EMNLP)*, pp. 1–8.
- 10816 Collins, M. (2013). Notes on natural language processing. <http://www.cs.columbia.edu/~mcollins/notes-spring2013.html>.
- 10817
- 10818 Collins, M. and T. Koo (2005). Discriminative reranking for natural language parsing.
10819 *Computational Linguistics* 31(1), 25–70.
- 10820 Collins, M. and B. Roark (2004). Incremental parsing with the perceptron algorithm. In
10821 *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pp.
10822 111. Association for Computational Linguistics.
- 10823 Collobert, R., K. Kavukcuoglu, and C. Farabet (2011). Torch7: A matlab-like environment
10824 for machine learning. Technical Report EPFL-CONF-192376, EPFL.
- 10825 Collobert, R. and J. Weston (2008). A unified architecture for natural language process-
10826 ing: Deep neural networks with multitask learning. In *Proceedings of the International*
10827 *Conference on Machine Learning (ICML)*, pp. 160–167.
- 10828 Collobert, R., J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa (2011). Nat-
10829 ural language processing (almost) from scratch. *Journal of Machine Learning Research* 12,
10830 2493–2537.
- 10831 Colton, S., J. Goodwin, and T. Veale (2012). Full-face poetry generation. In *Proceedings of*
10832 *the International Conference on Computational Creativity*, pp. 95–102.

- 10833 Conneau, A., D. Kiela, H. Schwenk, L. Barrault, and A. Bordes (2017). Supervised learning
10834 of universal sentence representations from natural language inference data. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 681–691.
10835
- 10836 Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2009). *Introduction to algorithms*
10837 (third ed.). MIT press.
- 10838 Cotterell, R., H. Schütze, and J. Eisner (2016). Morphological smoothing and extrapolation
10839 of word embeddings. In *Proceedings of the Association for Computational Linguistics (ACL)*,
10840 pp. 1651–1660.
- 10841 Coviello, L., Y. Sohn, A. D. Kramer, C. Marlow, M. Franceschetti, N. A. Christakis, and
10842 J. H. Fowler (2014). Detecting emotional contagion in massive social networks. *PloS
10843 one* 9(3), e90315.
- 10844 Covington, M. A. (2001). A fundamental algorithm for dependency parsing. In *Proceedings
10845 of the 39th annual ACM southeast conference*, pp. 95–102.
- 10846 Crammer, K., O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer (2006, December).
10847 Online passive-aggressive algorithms. *The Journal of Machine Learning Research* 7, 551–
10848 585.
- 10849 Crammer, K. and Y. Singer (2001). Pranking with ranking. In *Neural Information Processing
10850 Systems (NIPS)*, pp. 641–647.
- 10851 Creutz, M. and K. Lagus (2007). Unsupervised models for morpheme segmentation and
10852 morphology learning. *ACM Transactions on Speech and Language Processing (TSLP)* 4(1),
10853 3.
- 10854 Cross, J. and L. Huang (2016). Span-based constituency parsing with a structure-label
10855 system and provably optimal dynamic oracles. In *Proceedings of Empirical Methods for
10856 Natural Language Processing (EMNLP)*, pp. 1–11.
- 10857 Cucerzan, S. (2007). Large-scale named entity disambiguation based on wikipedia data.
10858 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 10859 Cui, H., R. Sun, K. Li, M.-Y. Kan, and T.-S. Chua (2005). Question answering passage
10860 retrieval using dependency relations. In *Proceedings of the 28th annual international ACM
10861 SIGIR conference on Research and development in information retrieval*, pp. 400–407. ACM.
- 10862 Cui, Y., Z. Chen, S. Wei, S. Wang, T. Liu, and G. Hu (2017). Attention-over-attention neural
10863 networks for reading comprehension. In *Proceedings of the Association for Computational
10864 Linguistics (ACL)*.

- 10865 Culotta, A. and J. Sorensen (2004). Dependency tree kernels for relation extraction. In
 10866 *Proceedings of the Association for Computational Linguistics (ACL)*.
- 10867 Culotta, A., M. Wick, and A. McCallum (2007). First-order probabilistic models for coref-
 10868 erence resolution. In *Proceedings of the North American Chapter of the Association for Com-*
 10869 *putational Linguistics (NAACL)*, pp. 81–88.
- 10870 Curry, H. B. and R. Feys (1958). *Combinatory Logic*, Volume I. Amsterdam: North Holland.
- 10871 Danescu-Niculescu-Mizil, C., M. Sudhof, D. Jurafsky, J. Leskovec, and C. Potts (2013). A
 10872 computational approach to politeness with application to social factors. In *Proceedings*
 10873 *of the Association for Computational Linguistics (ACL)*, pp. 250–259.
- 10874 Das, D., D. Chen, A. F. Martins, N. Schneider, and N. A. Smith (2014). Frame-semantic
 10875 parsing. *Computational Linguistics* 40(1), 9–56.
- 10876 Daumé III, H. (2007). Frustratingly easy domain adaptation. In *Proceedings of the Associa-*
 10877 *tion for Computational Linguistics (ACL)*.
- 10878 Daumé III, H., J. Langford, and D. Marcu (2009). Search-based structured prediction.
 10879 *Machine learning* 75(3), 297–325.
- 10880 Daumé III, H. and D. Marcu (2005). A large-scale exploration of effective global features
 10881 for a joint entity detection and tracking model. In *Proceedings of Empirical Methods for*
 10882 *Natural Language Processing (EMNLP)*, pp. 97–104.
- 10883 Dauphin, Y. N., R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio (2014). Identi-
 10884 fying and attacking the saddle point problem in high-dimensional non-convex opti-
 10885 mization. In *Neural Information Processing Systems (NIPS)*, pp. 2933–2941.
- 10886 Davidson, D. (1967). The logical form of action sentences. In N. Rescher (Ed.), *The Logic of*
 10887 *Decision and Action*. Pittsburgh: University of Pittsburgh Press.
- 10888 De Gispert, A. and J. B. Marino (2006). Catalan-english statistical machine translation
 10889 without parallel corpus: bridging through spanish. In *Proc. of 5th International Conference*
 10890 *on Language Resources and Evaluation (LREC)*, pp. 65–68. Citeseer.
- 10891 De Marneffe, M.-C. and C. D. Manning (2008). The stanford typed dependencies represen-
 10892 tation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain*
 10893 *Parser Evaluation*, pp. 1–8. Association for Computational Linguistics.
- 10894 Dean, J. and S. Ghemawat (2008). Mapreduce: simplified data processing on large clusters.
 10895 *Communications of the ACM* 51(1), 107–113.
- 10896 Deerwester, S. C., S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman (1990).
 10897 Indexing by latent semantic analysis. *JASIS* 41(6), 391–407.

- 10898 Dehdari, J. (2014). *A Neurophysiologically-Inspired Statistical Language Model*. Ph. D. thesis,
10899 The Ohio State University.
- 10900 Deisenroth, M. P., A. A. Faisal, and C. S. Ong (2018). *Mathematics For Machine Learning*.
10901 Cambridge UP.
- 10902 Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). Maximum likelihood from incom-
10903 plete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Method-
10904 ological)*, 1–38.
- 10905 Denis, P. and J. Baldridge (2007). A ranking approach to pronoun resolution. In *IJCAI*.
- 10906 Denis, P. and J. Baldridge (2008). Specialized models and ranking for coreference resolu-
10907 tion. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*,
10908 EMNLP '08, Stroudsburg, PA, USA, pp. 660–669. Association for Computational Lin-
10909 guistics.
- 10910 Denis, P. and J. Baldridge (2009). Global joint models for coreference resolution and named
10911 entity classification. *Procesamiento del Lenguaje Natural* 42.
- 10912 Derrida, J. (1985). Des tours de babel. In J. Graham (Ed.), *Difference in translation*. Ithaca,
10913 NY: Cornell University Press.
- 10914 Dhingra, B., H. Liu, Z. Yang, W. W. Cohen, and R. Salakhutdinov (2017). Gated-attention
10915 readers for text comprehension. In *Proceedings of the Association for Computational Lin-
10916 guistics (ACL)*.
- 10917 Diaconis, P. and B. Skyrms (2017). *Ten Great Ideas About Chance*. Princeton University
10918 Press.
- 10919 Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classifica-
10920 tion learning algorithms. *Neural computation* 10(7), 1895–1923.
- 10921 Dietterich, T. G., R. H. Lathrop, and T. Lozano-Pérez (1997). Solving the multiple instance
10922 problem with axis-parallel rectangles. *Artificial intelligence* 89(1), 31–71.
- 10923 Dimitrova, L., N. Ide, V. Petkevic, T. Erjavec, H. J. Kaalep, and D. Tufis (1998). Multext-
10924 east: Parallel and comparable corpora and lexicons for six central and eastern european
10925 languages. In *Proceedings of the 17th international conference on Computational linguistics-
10926 Volume 1*, pp. 315–319. Association for Computational Linguistics.
- 10927 Doddington, G. R., A. Mitchell, M. A. Przybocki, L. A. Ramshaw, S. Strassel, and R. M.
10928 Weischedel (2004). The automatic content extraction (ace) program-tasks, data, and
10929 evaluation. In *Proceedings of the Language Resources and Evaluation Conference*, pp. 837–
10930 840.

- 10931 dos Santos, C., B. Xiang, and B. Zhou (2015). Classifying relations by ranking with con-
 10932 volutional neural networks. In *Proceedings of the Association for Computational Linguistics*
 10933 (ACL), pp. 626–634.
- 10934 Dowty, D. (1991). Thematic proto-roles and argument selection. *Language*, 547–619.
- 10935 Dredze, M., P. McNamee, D. Rao, A. Gerber, and T. Finin (2010). Entity disambiguation
 10936 for knowledge base population. In *Proceedings of the 23rd International Conference on*
 10937 *Computational Linguistics*, pp. 277–285. Association for Computational Linguistics.
- 10938 Dredze, M., M. J. Paul, S. Bergsma, and H. Tran (2013). Carmen: A Twitter geolocation
 10939 system with applications to public health. In *AAAI workshop on expanding the boundaries*
 10940 *of health informatics using AI (HIAI)*, pp. 20–24.
- 10941 Dreyfus, H. L. (1992). *What computers still can't do: a critique of artificial reason*. MIT press.
- 10942 Du, L., W. Buntine, and M. Johnson (2013). Topic segmentation with a structured topic
 10943 model. In *Proceedings of the North American Chapter of the Association for Computational*
 10944 *Linguistics* (NAACL), pp. 190–200.
- 10945 Duchi, J., E. Hazan, and Y. Singer (2011). Adaptive subgradient methods for online learn-
 10946 ing and stochastic optimization. *The Journal of Machine Learning Research* 12, 2121–2159.
- 10947 Dunietz, J., L. Levin, and J. Carbonell (2017). The because corpus 2.0: Annotating causality
 10948 and overlapping relations. In *Proceedings of the Linguistic Annotation Workshop*.
- 10949 Durrett, G., T. Berg-Kirkpatrick, and D. Klein (2016). Learning-based single-document
 10950 summarization with compression and anaphoricity constraints. In *Proceedings of the*
 10951 *Association for Computational Linguistics* (ACL), pp. 1998–2008.
- 10952 Durrett, G. and D. Klein (2013). Easy victories and uphill battles in coreference resolution.
 10953 In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.
- 10954 Durrett, G. and D. Klein (2015). Neural crf parsing. In *Proceedings of the Association for*
 10955 *Computational Linguistics* (ACL).
- 10956 Dyer, C., M. Ballesteros, W. Ling, A. Matthews, and N. A. Smith (2015). Transition-based
 10957 dependency parsing with stack long short-term memory. In *Proceedings of the Association*
 10958 *for Computational Linguistics* (ACL), pp. 334–343.
- 10959 Dyer, C., A. Kuncoro, M. Ballesteros, and N. A. Smith (2016). Recurrent neural network
 10960 grammars. In *Proceedings of the North American Chapter of the Association for Computational*
 10961 *Linguistics* (NAACL), pp. 199–209.
- 10962 Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of*
 10963 *Standards B* 71(4), 233–240.

- 10964 Efron, B. and R. J. Tibshirani (1993). An introduction to the bootstrap: Monographs on
10965 statistics and applied probability, vol. 57. *New York and London: Chapman and Hall/CRC*.
- 10966 Eisenstein, J. (2009). Hierarchical text segmentation from multi-scale lexical cohesion. In
10967 *Proceedings of the North American Chapter of the Association for Computational Linguistics*
10968 (NAACL).
- 10969 Eisenstein, J. and R. Barzilay (2008). Bayesian unsupervised topic segmentation. In *Pro-*
10970 *ceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 10971 Eisner, J. (1997). State-of-the-art algorithms for minimum spanning trees: A tutorial dis-
10972 cussion.
- 10973 Eisner, J. (2000). Bilexical grammars and their cubic-time parsing algorithms. In *Advances*
10974 *in probabilistic and other parsing technologies*, pp. 29–61. Springer.
- 10975 Eisner, J. (2002). Parameter estimation for probabilistic finite-state transducers. In *Proced-*
10976 *ings of the Association for Computational Linguistics (ACL)*, pp. 1–8.
- 10977 Eisner, J. (2016). Inside-outside and forward-backward algorithms are just backprop. In
10978 *Proceedings of the Workshop on Structured Prediction for NLP*, pp. 1–17.
- 10979 Eisner, J. M. (1996). Three new probabilistic models for dependency parsing: An explo-
10980 ration. In *Proceedings of the International Conference on Computational Linguistics (COL-*
10981 *ING)*, pp. 340–345.
- 10982 Ekman, P. (1992). Are there basic emotions? *Psychological Review* 99(3), 550–553.
- 10983 Elman, J. L. (1990). Finding structure in time. *Cognitive science* 14(2), 179–211.
- 10984 Elman, J. L., E. A. Bates, M. H. Johnson, A. Karmiloff-Smith, D. Parisi, and K. Plunkett
10985 (1998). *Rethinking innateness: A connectionist perspective on development*, Volume 10. MIT
10986 press.
- 10987 Elsner, M. and E. Charniak (2010). Disentangling chat. *Computational Linguistics* 36(3),
10988 389–409.
- 10989 Esuli, A. and F. Sebastiani (2006). Sentiwordnet: A publicly available lexical resource for
10990 opinion mining. In *LREC*, Volume 6, pp. 417–422. Citeseer.
- 10991 Etzioni, O., A. Fader, J. Christensen, S. Soderland, and M. Mausam (2011). Open informa-
10992 tion extraction: The second generation. In *Proceedings of the International Joint Conference*
10993 *on Artificial Intelligence (IJCAI)*, pp. 3–10.

- 10994 Faruqui, M., J. Dodge, S. K. Jauhar, C. Dyer, E. Hovy, and N. A. Smith (2015). Retrofitting
 10995 word vectors to semantic lexicons. In *Proceedings of the North American Chapter of the*
 10996 *Association for Computational Linguistics (NAACL)*.
- 10997 Faruqui, M. and C. Dyer (2014). Improving vector space word representations using mul-
 10998 tilingual correlation. In *Proceedings of the European Chapter of the Association for Compu-*
 10999 *tational Linguistics (EACL)*, pp. 462–471.
- 11000 Faruqui, M., R. McDonald, and R. Soricut (2016). Morpho-syntactic lexicon generation
 11001 using graph-based semi-supervised learning. *Transactions of the Association for Compu-*
 11002 *tational Linguistics* 4, 1–16.
- 11003 Faruqui, M., Y. Tsvetkov, P. Rastogi, and C. Dyer (2016, August). Problems with evaluation
 11004 of word embeddings using word similarity tasks. In *Proceedings of the 1st Workshop on*
 11005 *Evaluating Vector-Space Representations for NLP*, Berlin, Germany, pp. 30–35. Association
 11006 for Computational Linguistics.
- 11007 Fellbaum, C. (2010). *WordNet*. Springer.
- 11008 Feng, V. W., Z. Lin, and G. Hirst (2014). The impact of deep hierarchical discourse struc-
 11009 tures in the evaluation of text coherence. In *Proceedings of the International Conference on*
 11010 *Computational Linguistics (COLING)*, pp. 940–949.
- 11011 Feng, X., L. Huang, D. Tang, H. Ji, B. Qin, and T. Liu (2016). A language-independent
 11012 neural network for event detection. In *Proceedings of the Association for Computational*
 11013 *Linguistics (ACL)*, pp. 66–71.
- 11014 Fernandes, E. R., C. N. dos Santos, and R. L. Milidiú (2014). Latent trees for coreference
 11015 resolution. *Computational Linguistics*.
- 11016 Ferrucci, D., E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W.
 11017 Murdock, E. Nyberg, J. Prager, et al. (2010). Building Watson: An overview of the
 11018 DeepQA project. *AI magazine* 31(3), 59–79.
- 11019 Ficler, J. and Y. Goldberg (2017, September). Controlling linguistic style aspects in neural
 11020 language generation. In *Proceedings of the Workshop on Stylistic Variation*, Copenhagen,
 11021 Denmark, pp. 94–104. Association for Computational Linguistics.
- 11022 Filippova, K. and M. Strube (2008). Sentence fusion via dependency graph compression.
 11023 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 177–
 11024 185.
- 11025 Fillmore, C. J. (1968). The case for case. In E. Bach and R. Harms (Eds.), *Universals in*
 11026 *linguistic theory*. Holt, Rinehart, and Winston.

- 11027 Fillmore, C. J. (1976). Frame semantics and the nature of language. *Annals of the New York
11028 Academy of Sciences* 280(1), 20–32.
- 11029 Fillmore, C. J. and C. Baker (2009). A frames approach to semantic analysis. In *The Oxford
11030 Handbook of Linguistic Analysis*. Oxford University Press.
- 11031 Finkel, J. R., T. Grenager, and C. Manning (2005). Incorporating non-local information
11032 into information extraction systems by gibbs sampling. In *Proceedings of the Association
11033 for Computational Linguistics (ACL)*, pp. 363–370.
- 11034 Finkel, J. R., T. Grenager, and C. D. Manning (2007). The infinite tree. In *Proceedings of the
11035 Association for Computational Linguistics (ACL)*, pp. 272–279.
- 11036 Finkel, J. R., A. Kleeman, and C. D. Manning (2008). Efficient, feature-based, conditional
11037 random field parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*,
11038 pp. 959–967.
- 11039 Finkel, J. R. and C. Manning (2009). Hierarchical bayesian domain adaptation. In *Proceed-
11040 ings of the North American Chapter of the Association for Computational Linguistics (NAACL)*,
11041 pp. 602–610.
- 11042 Finkel, J. R. and C. D. Manning (2008). Enforcing transitivity in coreference resolution.
11043 In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics
11044 on Human Language Technologies: Short Papers*, pp. 45–48. Association for Computational
11045 Linguistics.
- 11046 Finkelstein, L., E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin
11047 (2002). Placing search in context: The concept revisited. *ACM Transactions on Information
11048 Systems* 20(1), 116–131.
- 11049 Firth, J. R. (1957). *Papers in Linguistics 1934-1951*. Oxford University Press.
- 11050 Flanigan, J., S. Thomson, J. Carbonell, C. Dyer, and N. A. Smith (2014). A discrimina-
11051 tive graph-based parser for the abstract meaning representation. In *Proceedings of the
11052 Association for Computational Linguistics (ACL)*, pp. 1426–1436.
- 11053 Foltz, P. W., W. Kintsch, and T. K. Landauer (1998). The measurement of textual coherence
11054 with latent semantic analysis. *Discourse processes* 25(2-3), 285–307.
- 11055 Fordyce, C. S. (2007). Overview of the iwslt 2007 evaluation campaign. In *International
11056 Workshop on Spoken Language Translation (IWSLT) 2007*.
- 11057 Fox, H. (2002). Phrasal cohesion and statistical machine translation. In *Proceedings of
11058 Empirical Methods for Natural Language Processing (EMNLP)*, pp. 304–3111.

- 11059 Francis, W. and H. Kucera (1982). *Frequency analysis of English usage*. Houghton Mifflin
11060 Company.
- 11061 Francis, W. N. (1964). A standard sample of present-day English for use with digital
11062 computers. Report to the U.S Office of Education on Cooperative Research Project No.
11063 E-007.
- 11064 Freund, Y. and R. E. Schapire (1999). Large margin classification using the perceptron
11065 algorithm. *Machine learning* 37(3), 277–296.
- 11066 Fromkin, V., R. Rodman, and N. Hyams (2013). *An introduction to language*. Cengage
11067 Learning.
- 11068 Fundel, K., R. Küffner, and R. Zimmer (2007). Relex – relation extraction using depen-
11069 dency parse trees. *Bioinformatics* 23(3), 365–371.
- 11070 Gabow, H. N., Z. Galil, T. Spencer, and R. E. Tarjan (1986). Efficient algorithms for finding
11071 minimum spanning trees in undirected and directed graphs. *Combinatorica* 6(2), 109–
11072 122.
- 11073 Gabrilovich, E. and S. Markovitch (2007). Computing semantic relatedness using
11074 wikipedia-based explicit semantic analysis. In *Proceedings of the International Joint Con-
11075 ference on Artificial Intelligence (IJCAI)*, Volume 7, pp. 1606–1611.
- 11076 Gage, P. (1994). A new algorithm for data compression. *The C Users Journal* 12(2), 23–38.
- 11077 Gale, W. A., K. W. Church, and D. Yarowsky (1992). One sense per discourse. In *Pro-
11078 ceedings of the workshop on Speech and Natural Language*, pp. 233–237. Association for
11079 Computational Linguistics.
- 11080 Galley, M., M. Hopkins, K. Knight, and D. Marcu (2004). What's in a translation rule? In
11081 *Proceedings of the North American Chapter of the Association for Computational Linguistics*
11082 (NAACL), pp. 273–280.
- 11083 Galley, M., K. R. McKeown, E. Fosler-Lussier, and H. Jing (2003). Discourse segmentation
11084 of multi-party conversation. In *Proceedings of the Association for Computational Linguistics*
11085 (ACL).
- 11086 Ganchev, K. and M. Dredze (2008). Small statistical models by random feature mixing. In
11087 *Proceedings of the ACL08 HLT Workshop on Mobile Language Processing*, pp. 19–20.
- 11088 Ganchev, K., J. Graça, J. Gillenwater, and B. Taskar (2010). Posterior regularization for
11089 structured latent variable models. *The Journal of Machine Learning Research* 11, 2001–
11090 2049.

- 11091 Ganin, Y., E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand,
11092 and V. Lempitsky (2016). Domain-adversarial training of neural networks. *Journal of
11093 Machine Learning Research* 17(59), 1–35.
- 11094 Gao, J., G. Andrew, M. Johnson, and K. Toutanova (2007). A comparative study of param-
11095 eter estimation methods for statistical natural language processing. In *Proceedings of the
11096 Association for Computational Linguistics (ACL)*, pp. 824–831.
- 11097 Gatt, A. and E. Krahmer (2018). Survey of the state of the art in natural language genera-
11098 tion: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research* 61,
11099 65–170.
- 11100 Gatt, A. and E. Reiter (2009). Simplenlg: A realisation engine for practical applications.
11101 In *Proceedings of the 12th European Workshop on Natural Language Generation*, pp. 90–93.
11102 Association for Computational Linguistics.
- 11103 Ge, D., X. Jiang, and Y. Ye (2011). A note on the complexity of $l_1 p$ minimization. *Mathe-
11104 matical programming* 129(2), 285–299.
- 11105 Ge, N., J. Hale, and E. Charniak (1998). A statistical approach to anaphora resolution. In
11106 *Proceedings of the sixth workshop on very large corpora*, Volume 71, pp. 76.
- 11107 Ge, R., F. Huang, C. Jin, and Y. Yuan (2015). Escaping from saddle points — online stochas-
11108 tic gradient for tensor decomposition. In P. Grünwald, E. Hazan, and S. Kale (Eds.),
11109 *Proceedings of the Conference On Learning Theory (COLT)*.
- 11110 Ge, R. and R. J. Mooney (2005). A statistical semantic parser that integrates syntax and
11111 semantics. In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp.
11112 9–16.
- 11113 Geach, P. T. (1962). *Reference and generality: An examination of some medieval and modern
11114 theories*. Cornell University Press.
- 11115 Gildea, D. and D. Jurafsky (2002). Automatic labeling of semantic roles. *Computational
11116 linguistics* 28(3), 245–288.
- 11117 Gimpel, K., N. Schneider, B. O’Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yo-
11118 gatama, J. Flanigan, and N. A. Smith (2011). Part-of-speech tagging for Twitter: an-
11119 notation, features, and experiments. In *Proceedings of the Association for Computational
11120 Linguistics (ACL)*, pp. 42–47.
- 11121 Glass, J., T. J. Hazen, S. Cyphers, I. Malioutov, D. Huynh, and R. Barzilay (2007). Recent
11122 progress in the mit spoken lecture processing project. In *Eighth Annual Conference of the
11123 International Speech Communication Association*.

- 11124 Glorot, X. and Y. Bengio (2010). Understanding the difficulty of training deep feedforward
 11125 neural networks. In *Proceedings of Artificial Intelligence and Statistics (AISTATS)*, pp. 249–
 11126 256.
- 11127 Glorot, X., A. Bordes, and Y. Bengio (2011). Deep sparse rectifier networks. In *Proceedings*
 11128 *of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP*
 11129 *Volume*, Volume 15, pp. 315–323.
- 11130 Godfrey, J. J., E. C. Holliman, and J. McDaniel (1992). Switchboard: Telephone speech
 11131 corpus for research and development. In *Proceedings of the International Conference on*
 11132 *Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 517–520. IEEE.
- 11133 Goldberg, Y. (2017a, June). An adversarial review of “adversarial generation of
 11134 natural language”. [https://medium.com/@yoav.goldberg/an-adversarial-review-of-](https://medium.com/@yoav.goldberg/an-adversarial-review-of-adversarial-generation-of-natural-language-409ac3378bd7)
 11135 [adversarial-generation-of-natural-language-409ac3378bd7](https://medium.com/@yoav.goldberg/an-adversarial-review-of-adversarial-generation-of-natural-language-409ac3378bd7).
- 11136 Goldberg, Y. (2017b). *Neural Network Methods for Natural Language Processing*. Synthesis
 11137 Lectures on Human Language Technologies. Morgan & Claypool Publishers.
- 11138 Goldberg, Y. and M. Elhadad (2010). An efficient algorithm for easy-first non-directional
 11139 dependency parsing. In *Proceedings of the North American Chapter of the Association for*
 11140 *Computational Linguistics (NAACL)*, pp. 742–750.
- 11141 Goldberg, Y. and J. Nivre (2012). A dynamic oracle for arc-eager dependency parsing.
 11142 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.
 11143 959–976.
- 11144 Goldberg, Y., K. Zhao, and L. Huang (2013). Efficient implementation of beam-search
 11145 incremental parsers. In *ACL (2)*, pp. 628–633.
- 11146 Goldwater, S. and T. Griffiths (2007). A fully bayesian approach to unsupervised part-of-
 11147 speech tagging. In *Annual meeting-association for computational linguistics*, Volume 45.
- 11148 Gonçalo Oliveira, H. R., F. A. Cardoso, and F. C. Pereira (2007). Tra-la-lyrics: An approach
 11149 to generate text based on rhythm. In *Proceedings of the 4th. International Joint Workshop*
 11150 *on Computational Creativity*. A. Cardoso and G. Wiggins.
- 11151 Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep learning*. MIT Press.
- 11152 Goodman, J. T. (2001). A bit of progress in language modeling. *Computer Speech & Lan-*
 11153 *guage* 15(4), 403–434.
- 11154 Gouws, S., D. Metzler, C. Cai, and E. Hovy (2011). Contextual bearing on linguistic varia-
 11155 tion in social media. In *LASM*.

- 11156 Goyal, A., H. Daume III, and S. Venkatasubramanian (2009). Streaming for large scale
11157 nlp: Language modeling. In *Proceedings of the North American Chapter of the Association*
11158 for Computational Linguistics (NAACL), pp. 512–520.
- 11159 Graves, A. (2012). Sequence transduction with recurrent neural networks. In *Proceedings*
11160 of the International Conference on Machine Learning (ICML).
- 11161 Graves, A. and N. Jaitly (2014). Towards end-to-end speech recognition with recur-
11162 rent neural networks. In *Proceedings of the International Conference on Machine Learning*
11163 (ICML), pp. 1764–1772.
- 11164 Graves, A. and J. Schmidhuber (2005). Framewise phoneme classification with bidirec-
11165 tional lstm and other neural network architectures. *Neural Networks* 18(5), 602–610.
- 11166 Grice, H. P. (1975). Logic and conversation. In P. Cole and J. L. Morgan (Eds.), *Syntax and*
11167 *Semantics Volume 3: Speech Acts*, pp. 41–58. Academic Press.
- 11168 Grishman, R. (2012). Information extraction: Capabilities and challenges. Notes prepared
11169 for the 2012 International Winter School in Language and Speech Technologies, Rovira
11170 i Virgili University, Tarragona, Spain.
- 11171 Grishman, R. (2015). Information extraction. *IEEE Intelligent Systems* 30(5), 8–15.
- 11172 Grishman, R., C. Macleod, and J. Sterling (1992). Evaluating parsing strategies using
11173 standardized parse files. In *Proceedings of the third conference on Applied natural language*
11174 *processing*, pp. 156–161. Association for Computational Linguistics.
- 11175 Grishman, R. and B. Sundheim (1996). Message understanding conference-6: A brief his-
11176 tory. In *Proceedings of the International Conference on Computational Linguistics (COLING)*,
11177 pp. 466–471.
- 11178 Groenendijk, J. and M. Stokhof (1991). Dynamic predicate logic. *Linguistics and philoso-*
11179 *phy* 14(1), 39–100.
- 11180 Grosz, B. J. (1979). Focusing and description in natural language dialogues. Technical
11181 report, SRI INTERNATIONAL MENLO PARK CA.
- 11182 Grosz, B. J., S. Weinstein, and A. K. Joshi (1995). Centering: A framework for modeling
11183 the local coherence of discourse. *Computational linguistics* 21(2), 203–225.
- 11184 Gu, J., Z. Lu, H. Li, and V. O. Li (2016). Incorporating copying mechanism in sequence-to-
11185 sequence learning. In *Proceedings of the Association for Computational Linguistics (ACL)*,
11186 pp. 1631–1640.
- 11187 Gulcehre, C., S. Ahn, R. Nallapati, B. Zhou, and Y. Bengio (2016). Pointing the unknown
11188 words. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 140–149.

- 11189 Gutmann, M. U. and A. Hyvärinen (2012). Noise-contrastive estimation of unnormalized
 11190 statistical models, with applications to natural image statistics. *The Journal of Machine
 11191 Learning Research* 13(1), 307–361.
- 11192 Haghghi, A. and D. Klein (2007). Unsupervised coreference resolution in a nonparametric
 11193 bayesian model. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11194 Haghghi, A. and D. Klein (2009). Simple coreference resolution with rich syntactic and
 11195 semantic features. In *Proceedings of Empirical Methods for Natural Language Processing
 11196 (EMNLP)*, pp. 1152–1161.
- 11197 Haghghi, A. and D. Klein (2010). Coreference resolution in a modular, entity-centered
 11198 model. In *Proceedings of the North American Chapter of the Association for Computational
 11199 Linguistics (NAACL)*, pp. 385–393.
- 11200 Hajič, J. and B. Hladká (1998). Tagging inflective languages: Prediction of morphological
 11201 categories for a rich, structured tagset. In *Proceedings of the Association for Computational
 11202 Linguistics (ACL)*, pp. 483–490.
- 11203 Halliday, M. and R. Hasan (1976). *Cohesion in English*. London: Longman.
- 11204 Hammerton, J. (2003). Named entity recognition with long short-term memory. In *Pro-
 11205 ceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 172–175.
- 11206 Han, X. and L. Sun (2012). An entity-topic model for entity linking. In *Proceedings of
 11207 Empirical Methods for Natural Language Processing (EMNLP)*, pp. 105–115.
- 11208 Han, X., L. Sun, and J. Zhao (2011). Collective entity linking in web text: a graph-based
 11209 method. In *Proceedings of ACM SIGIR conference on Research and development in informa-
 11210 tion retrieval*, pp. 765–774.
- 11211 Hannak, A., E. Anderson, L. F. Barrett, S. Lehmann, A. Mislove, and M. Riedewald (2012).
 11212 Tweetin'in the rain: Exploring societal-scale effects of weather on mood. In *Proceedings
 11213 of the International Conference on Web and Social Media (ICWSM)*.
- 11214 Hardmeier, C. (2012). Discourse in statistical machine translation. a survey and a case
 11215 study. *Discours. Revue de linguistique, psycholinguistique et informatique. A journal of lin-
 11216 guistics, psycholinguistics and computational linguistics* (11).
- 11217 Haspelmath, M. and A. Sims (2013). *Understanding morphology*. Routledge.
- 11218 Hastie, T., R. Tibshirani, and J. Friedman (2009). *The elements of statistical learning* (Second
 11219 ed.). New York: Springer.

- 11220 Hatzivassiloglou, V. and K. R. McKeown (1997). Predicting the semantic orientation of
11221 adjectives. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 174–
11222 181.
- 11223 Hayes, A. F. and K. Krippendorff (2007). Answering the call for a standard reliability
11224 measure for coding data. *Communication methods and measures* 1(1), 77–89.
- 11225 He, H., A. Balakrishnan, M. Eric, and P. Liang (2017). Learning symmetric collaborative
11226 dialogue agents with dynamic knowledge graph embeddings. In *Proceedings of the As-
11227 sociation for Computational Linguistics (ACL)*, pp. 1766–1776.
- 11228 He, K., X. Zhang, S. Ren, and J. Sun (2015). Delving deep into rectifiers: Surpassing
11229 human-level performance on imagenet classification. In *Proceedings of the International
11230 Conference on Computer Vision (ICCV)*, pp. 1026–1034.
- 11231 He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep residual learning for image recognition.
11232 In *Proceedings of the International Conference on Computer Vision (ICCV)*, pp. 770–778.
- 11233 He, L., K. Lee, M. Lewis, and L. Zettlemoyer (2017). Deep semantic role labeling: What
11234 works and what's next. In *Proceedings of the Association for Computational Linguistics
11235 (ACL)*.
- 11236 He, Z., S. Liu, M. Li, M. Zhou, L. Zhang, and H. Wang (2013). Learning entity repre-
11237 sentation for entity disambiguation. In *Proceedings of the Association for Computational
11238 Linguistics (ACL)*, pp. 30–34.
- 11239 Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. In
11240 *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp. 539–
11241 545. Association for Computational Linguistics.
- 11242 Hearst, M. A. (1997). Texttiling: Segmenting text into multi-paragraph subtopic passages.
11243 *Computational linguistics* 23(1), 33–64.
- 11244 Heerschap, B., F. Goossen, A. Hogenboom, F. Frasincar, U. Kaymak, and F. de Jong (2011).
11245 Polarity analysis of texts using discourse structure. In *Proceedings of the 20th ACM inter-
11246 national conference on Information and knowledge management*, pp. 1061–1070. ACM.
- 11247 Henderson, J. (2004). Discriminative training of a neural network statistical parser. In
11248 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 95–102.
- 11249 Hendrickx, I., S. N. Kim, Z. Kozareva, P. Nakov, D. Ó Séaghdha, S. Padó, M. Pennacchiotti,
11250 L. Romano, and S. Szpakowicz (2009). SemEval-2010 task 8: Multi-way classification of
11251 semantic relations between pairs of nominals. In *Proceedings of the Workshop on Semantic
11252 Evaluations: Recent Achievements and Future Directions*, pp. 94–99. Association for Com-
11253 putational Linguistics.

- 11254 Hermann, K. M., T. Kočiský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and
 11255 P. Blunsom (2015). Teaching machines to read and comprehend. In *Advances in Neu-*
 11256 *ral Information Processing Systems*, pp. 1693–1701.
- 11257 Hernault, H., H. Prendinger, D. A. duVerle, and M. Ishizuka (2010). HILDA: A discourse
 11258 parser using support vector machine classification. *Dialogue and Discourse* 1(3), 1–33.
- 11259 Hill, F., A. Bordes, S. Chopra, and J. Weston (2016). The goldilocks principle: Reading
 11260 children’s books with explicit memory representations. In *Proceedings of the International*
 11261 *Conference on Learning Representations (ICLR)*.
- 11262 Hindle, D. and M. Rooth (1993). Structural ambiguity and lexical relations. *Computational*
 11263 *linguistics* 19(1), 103–120.
- 11264 Hirao, T., Y. Yoshida, M. Nishino, N. Yasuda, and M. Nagata (2013). Single-document
 11265 summarization as a tree knapsack problem. In *Proceedings of Empirical Methods for Nat-*
 11266 *ural Language Processing (EMNLP)*, pp. 1515–1520.
- 11267 Hirschman, L. and R. Gaizauskas (2001). Natural language question answering: the view
 11268 from here. *natural language engineering* 7(4), 275–300.
- 11269 Hirschman, L., M. Light, E. Breck, and J. D. Burger (1999). Deep read: A reading compre-
 11270 hension system. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
 11271 325–332.
- 11272 Hobbs, J. R. (1978). Resolving pronoun references. *Lingua* 44(4), 311–338.
- 11273 Hobbs, J. R., D. Appelt, J. Bear, D. Israel, M. Kameyama, M. Stickel, and M. Tyson (1997).
 11274 Fastus: A cascaded finite-state transducer for extracting information from natural-
 11275 language text. *Finite-state language processing*, 383–406.
- 11276 Hochreiter, S. and J. Schmidhuber (1997). Long short-term memory. *Neural computa-*
 11277 *tion* 9(8), 1735–1780.
- 11278 Hockenmaier, J. and M. Steedman (2007). Ccgbank: a corpus of ccg derivations and de-
 11279 pendency structures extracted from the penn treebank. *Computational Linguistics* 33(3),
 11280 355–396.
- 11281 Hoffart, J., M. A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, B. Taneva,
 11282 S. Thater, and G. Weikum (2011). Robust disambiguation of named entities in text. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 782–792.
- 11284 Hoffmann, R., C. Zhang, X. Ling, L. Zettlemoyer, and D. S. Weld (2011). Knowledge-based
 11285 weak supervision for information extraction of overlapping relations. In *Proceedings of*
 11286 *the Association for Computational Linguistics (ACL)*, pp. 541–550.

- 11287 Holmstrom, L. and P. Koistinen (1992). Using additive noise in back-propagation training.
11288 *IEEE Transactions on Neural Networks* 3(1), 24–38.
- 11289 Hovy, E. and J. Lavid (2010). Towards a ‘science’ of corpus annotation: a new method-
11290 ological challenge for corpus linguistics. *International journal of translation* 22(1), 13–36.
- 11291 Hsu, D., S. M. Kakade, and T. Zhang (2012). A spectral algorithm for learning hidden
11292 markov models. *Journal of Computer and System Sciences* 78(5), 1460–1480.
- 11293 Hu, M. and B. Liu (2004). Mining and summarizing customer reviews. In *Proceedings of*
11294 *Knowledge Discovery and Data Mining (KDD)*, pp. 168–177.
- 11295 Hu, Z., Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing (2017). Toward controlled
11296 generation of text. In *International Conference on Machine Learning*, pp. 1587–1596.
- 11297 Huang, F. and A. Yates (2012). Biased representation learning for domain adaptation. In
11298 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1313–1323.
- 11299 Huang, L., S. Fayong, and Y. Guo (2012). Structured perceptron with inexact search. In
11300 *Proceedings of the North American Chapter of the Association for Computational Linguistics*
11301 (NAACL), pp. 142–151.
- 11302 Huang, Y. (2015). *Pragmatics* (Second ed.). Oxford Textbooks in Linguistics. Oxford Uni-
11303 versity Press.
- 11304 Huang, Z., W. Xu, and K. Yu (2015). Bidirectional lstm-crf models for sequence tagging.
11305 *arXiv preprint arXiv:1508.01991*.
- 11306 Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes.
11307 *Proceedings of the IRE* 40(9), 1098–1101.
- 11308 Humphreys, K., R. Gaizauskas, and S. Azzam (1997). Event coreference for information
11309 extraction. In *Proceedings of a Workshop on Operational Factors in Practical, Robust Anaphora*
11310 *Resolution for Unrestricted Texts*, pp. 75–81. Association for Computational Linguistics.
- 11311 Ide, N. and Y. Wilks (2006). Making sense about sense. In *Word sense disambiguation*, pp.
11312 47–73. Springer.
- 11313 Ioffe, S. and C. Szegedy (2015). Batch normalization: Accelerating deep network train-
11314 ing by reducing internal covariate shift. In *Proceedings of the International Conference on*
11315 *Machine Learning (ICML)*, pp. 448–456.
- 11316 Isozaki, H., T. Hirao, K. Duh, K. Sudoh, and H. Tsukada (2010). Automatic evaluation
11317 of translation quality for distant language pairs. In *Proceedings of Empirical Methods for*
11318 *Natural Language Processing (EMNLP)*, pp. 944–952.

- 11319 Iyyer, M., V. Manjunatha, J. Boyd-Graber, and H. Daumé III (2015). Deep unordered com-
 11320 position rivals syntactic methods for text classification. In *Proceedings of the Association
 11321 for Computational Linguistics (ACL)*, pp. 1681–1691.
- 11322 James, G., D. Witten, T. Hastie, and R. Tibshirani (2013). *An introduction to statistical learn-
 11323 ing*, Volume 112. Springer.
- 11324 Janin, A., D. Baron, J. Edwards, D. Ellis, D. Gelbart, N. Morgan, B. Peskin, T. Pfau,
 11325 E. Shriberg, A. Stolcke, et al. (2003). The ICSI meeting corpus. In *Acoustics, Speech,
 11326 and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference
 11327 on*, Volume 1, pp. I–I. IEEE.
- 11328 Jean, S., K. Cho, R. Memisevic, and Y. Bengio (2015). On using very large target vocab-
 11329 ular for neural machine translation. In *Proceedings of the Association for Computational
 11330 Linguistics (ACL)*, pp. 1–10.
- 11331 Jeong, M., C.-Y. Lin, and G. G. Lee (2009). Semi-supervised speech act recognition in
 11332 emails and forums. In *Proceedings of Empirical Methods for Natural Language Processing
 11333 (EMNLP)*, pp. 1250–1259.
- 11334 Ji, H. and R. Grishman (2011). Knowledge base population: Successful approaches and
 11335 challenges. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1148–
 11336 1158.
- 11337 Ji, Y., T. Cohn, L. Kong, C. Dyer, and J. Eisenstein (2015). Document context language
 11338 models. In *International Conference on Learning Representations, Workshop Track*, Volume
 11339 abs/1511.03962.
- 11340 Ji, Y. and J. Eisenstein (2014). Representation learning for text-level discourse parsing. In
 11341 *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11342 Ji, Y. and J. Eisenstein (2015, June). One vector is not enough: Entity-augmented distribu-
 11343 tional semantics for discourse relations. *Transactions of the Association for Computational
 11344 Linguistics (TACL)*.
- 11345 Ji, Y., G. Haffari, and J. Eisenstein (2016). A latent variable recurrent neural network for
 11346 discourse relation language models. In *Proceedings of the North American Chapter of the
 11347 Association for Computational Linguistics (NAACL)*.
- 11348 Ji, Y. and N. A. Smith (2017). Neural discourse structure for text categorization. In *Pro-
 11349 ceedings of the Association for Computational Linguistics (ACL)*, pp. 996–1005.
- 11350 Ji, Y., C. Tan, S. Martschat, Y. Choi, and N. A. Smith (2017). Dynamic entity representations
 11351 in neural language models. In *Proceedings of Empirical Methods for Natural Language
 11352 Processing (EMNLP)*, pp. 1831–1840.

- 11353 Jiang, L., M. Yu, M. Zhou, X. Liu, and T. Zhao (2011). Target-dependent twitter sentiment
11354 classification. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
11355 151–160.
- 11356 Jing, H. (2000). Sentence reduction for automatic text summarization. In *Proceedings of
11357 the sixth conference on Applied natural language processing*, pp. 310–315. Association for
11358 Computational Linguistics.
- 11359 Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of
11360 Knowledge Discovery and Data Mining (KDD)*, pp. 133–142.
- 11361 Jockers, M. L. (2015). Szuzhet? <http://bla.bla.com>.
- 11362 Johnson, A. E., T. J. Pollard, L. Shen, H. L. Li-wei, M. Feng, M. Ghassemi, B. Moody,
11363 P. Szolovits, L. A. Celi, and R. G. Mark (2016). Mimic-iii, a freely accessible critical care
11364 database. *Scientific data* 3, 160035.
- 11365 Johnson, M. (1998). Pcfg models of linguistic tree representations. *Computational Linguis-
11366 tics* 24(4), 613–632.
- 11367 Johnson, R. and T. Zhang (2017). Deep pyramid convolutional neural networks for text
11368 categorization. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
11369 562–570.
- 11370 Joshi, A. K. (1985). How much context-sensitivity is required to provide reasonable struc-
11371 tural descriptions? – tree adjoining grammars. In *Natural Language Processing – Theoret-
11372 ical, Computational and Psychological Perspective*. New York, NY: Cambridge University
11373 Press.
- 11374 Joshi, A. K. and Y. Schabes (1997). Tree-adjoining grammars. In *Handbook of formal lan-
11375 guages*, pp. 69–123. Springer.
- 11376 Joshi, A. K., K. V. Shanker, and D. Weir (1991). The convergence of mildly context-sensitive
11377 grammar formalisms. In *Foundational Issues in Natural Language Processing*. Cambridge
11378 MA: MIT Press.
- 11379 Jozefowicz, R., O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu (2016). Exploring the limits
11380 of language modeling. *arXiv preprint arXiv:1602.02410*.
- 11381 Jozefowicz, R., W. Zaremba, and I. Sutskever (2015). An empirical exploration of recurrent
11382 network architectures. In *Proceedings of the International Conference on Machine Learning
11383 (ICML)*, pp. 2342–2350.
- 11384 Jurafsky, D. (1996). A probabilistic model of lexical and syntactic access and disambigua-
11385 tion. *Cognitive Science* 20(2), 137–194.

- 11386 Jurafsky, D. and J. H. Martin (2009). *Speech and Language Processing* (Second ed.). Prentice
11387 Hall.
- 11388 Jurafsky, D. and J. H. Martin (2018). *Speech and Language Processing* (Third ed.). Prentice
11389 Hall.
- 11390 Kadlec, R., M. Schmid, O. Bajgar, and J. Kleindienst (2016). Text understanding with
11391 the attention sum reader network. In *Proceedings of the Association for Computational
11392 Linguistics (ACL)*, pp. 908–918.
- 11393 Kalchbrenner, N. and P. Blunsom (2013, August). Recurrent convolutional neural net-
11394 works for discourse compositionality. In *Proceedings of the Workshop on Continuous Vec-
11395 tor Space Models and their Compositionality*, Sofia, Bulgaria, pp. 119–126. Association for
11396 Computational Linguistics.
- 11397 Kalchbrenner, N., E. Grefenstette, and P. Blunsom (2014). A convolutional neural network
11398 for modelling sentences. In *Proceedings of the Association for Computational Linguistics
(ACL)*, pp. 655–665.
- 11400 Karlsson, F. (2007). Constraints on multiple center-embedding of clauses. *Journal of Lin-
11401 guistics* 43(02), 365–392.
- 11402 Kate, R. J., Y. W. Wong, and R. J. Mooney (2005). Learning to transform natural to formal
11403 languages. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.
- 11404 Kehler, A. (2007). Rethinking the SMASH approach to pronoun interpretation. In *Interdis-
11405 ciplinary perspectives on reference processing*, New Directions in Cognitive Science Series,
11406 pp. 95–122. Oxford University Press.
- 11407 Kibble, R. and R. Power (2004). Optimizing referential coherence in text generation. *Com-
11408 putational Linguistics* 30(4), 401–416.
- 11409 Kilgarriff, A. (1997). I don't believe in word senses. *Computers and the Humanities* 31(2),
11410 91–113.
- 11411 Kilgarriff, A. and G. Grefenstette (2003). Introduction to the special issue on the web as
11412 corpus. *Computational linguistics* 29(3), 333–347.
- 11413 Kim, M.-J. (2002). Does korean have adjectives? *MIT Working Papers in Linguistics* 43,
11414 71–89.
- 11415 Kim, S.-M. and E. Hovy (2006, July). Extracting opinions, opinion holders, and topics
11416 expressed in online news media text. In *Proceedings of the Workshop on Sentiment and
11417 Subjectivity in Text*, Sydney, Australia, pp. 1–8. Association for Computational Linguis-
11418 tics.

- 11419 Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings*
11420 *of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1746–1751.
- 11421 Kim, Y., C. Denton, L. Hoang, and A. M. Rush (2017). Structured attention networks. In
11422 *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 11423 Kim, Y., Y. Jernite, D. Sontag, and A. M. Rush (2016). Character-aware neural language
11424 models. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.
- 11425 Kingma, D. and J. Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint*
11426 *arXiv:1412.6980*.
- 11427 Kingma, D. P. and M. Welling (2014). Auto-encoding variational bayes. In *Proceedings of*
11428 *the International Conference on Learning Representations (ICLR)*.
- 11429 Kiperwasser, E. and Y. Goldberg (2016). Simple and accurate dependency parsing using
11430 bidirectional lstm feature representations. *Transactions of the Association for Compu-*
11431 *tational Linguistics* 4, 313–327.
- 11432 Kipper-Schuler, K. (2005). *VerbNet: A broad-coverage, comprehensive verb lexicon*. Ph. D.
11433 thesis, Computer and Information Science, University of Pennsylvania.
- 11434 Kiros, R., R. Salakhutdinov, and R. Zemel (2014). Multimodal neural language models. In
11435 *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 595–603.
- 11436 Kiros, R., Y. Zhu, R. Salakhudinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler
11437 (2015). Skip-thought vectors. In *Neural Information Processing Systems (NIPS)*.
- 11438 Klein, D. and C. D. Manning (2003). Accurate unlexicalized parsing. In *Proceedings of the*
11439 *Association for Computational Linguistics (ACL)*, pp. 423–430.
- 11440 Klein, D. and C. D. Manning (2004). Corpus-based induction of syntactic structure: Mod-
11441 ells of dependency and constituency. In *Proceedings of the Association for Computational*
11442 *Linguistics (ACL)*.
- 11443 Klein, G., Y. Kim, Y. Deng, J. Senellart, and A. M. Rush (2017). Opennmt: Open-source
11444 toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*.
- 11445 Klementiev, A., I. Titov, and B. Bhattachari (2012). Inducing crosslingual distributed repre-
11446 sentations of words. In *Proceedings of the International Conference on Computational Lin-*
11447 *guistics (COLING)*, pp. 1459–1474.
- 11448 Klenner, M. (2007). Enforcing consistency on coreference sets. In *Recent Advances in Natu-*
11449 *ral Language Processing (RANLP)*, pp. 323–328.

- 11450 Knight, K. (1999). Decoding complexity in word-replacement translation models. *Computational Linguistics* 25(4), 607–615.
- 11451
- 11452 Knight, K. and J. Graehl (1998). Machine transliteration. *Computational Linguistics* 24(4), 599–612.
- 11453
- 11454 Knight, K. and D. Marcu (2000). Statistics-based summarization-step one: Sentence compression. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 703–710.
- 11455
- 11456
- 11457 Knight, K. and J. May (2009). Applications of weighted automata in natural language processing. In *Handbook of Weighted Automata*, pp. 571–596. Springer.
- 11458
- 11459 Knott, A. (1996). *A data-driven methodology for motivating a set of coherence relations*. Ph. D. thesis, The University of Edinburgh.
- 11460
- 11461 Koehn, P. (2009). *Statistical machine translation*. Cambridge University Press.
- 11462 Koehn, P. (2017). Neural machine translation. *arXiv preprint arXiv:1709.07809*.
- 11463 Konstas, I. and M. Lapata (2013). A global model for concept-to-text generation. *Journal of Artificial Intelligence Research* 48, 305–346.
- 11464
- 11465 Koo, T., X. Carreras, and M. Collins (2008, jun). Simple semi-supervised dependency parsing. In *Proceedings of ACL-08: HLT*, Columbus, Ohio, pp. 595–603. Association for Computational Linguistics.
- 11466
- 11467
- 11468 Koo, T. and M. Collins (2005). Hidden-variable models for discriminative reranking. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 507–514.
- 11469
- 11470 Koo, T. and M. Collins (2010). Efficient third-order dependency parsers. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11471
- 11472 Koo, T., A. Globerson, X. Carreras, and M. Collins (2007). Structured prediction models via the matrix-tree theorem. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 141–150.
- 11473
- 11474
- 11475 Kovach, B. and T. Rosenstiel (2014). *The elements of journalism: What newspeople should know and the public should expect*. Three Rivers Press.
- 11476
- 11477 Krishnamurthy, J. (2016). Probabilistic models for learning a semantic parser lexicon. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 606–616.
- 11478
- 11479

- 11480 Krishnamurthy, J. and T. M. Mitchell (2012). Weakly supervised training of semantic
11481 parsers. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
11482 pp. 754–765.
- 11483 Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep
11484 convolutional neural networks. In *Neural Information Processing Systems (NIPS)*, pp.
11485 1097–1105.
- 11486 Kübler, S., R. McDonald, and J. Nivre (2009). Dependency parsing. *Synthesis Lectures on*
11487 *Human Language Technologies* 1(1), 1–127.
- 11488 Kuhlmann, M. and J. Nivre (2010). Transition-based techniques for non-projective depen-
11489 dependency parsing. *Northern European Journal of Language Technology (NEJLT)* 2(1), 1–19.
- 11490 Kummerfeld, J. K., T. Berg-Kirkpatrick, and D. Klein (2015). An empirical analysis of op-
11491 timization for max-margin NLP. In *Proceedings of Empirical Methods for Natural Language*
11492 *Processing (EMNLP)*.
- 11493 Kwiatkowski, T., S. Goldwater, L. Zettlemoyer, and M. Steedman (2012). A probabilistic
11494 model of syntactic and semantic acquisition from child-directed utterances and their
11495 meanings. In *Proceedings of the European Chapter of the Association for Computational Lin-*
11496 *guistics (EACL)*, pp. 234–244.
- 11497 Lafferty, J., A. McCallum, and F. Pereira (2001). Conditional random fields: Probabilistic
11498 models for segmenting and labeling sequence data. In *icml*.
- 11499 Lakoff, G. (1973). Hedges: A study in meaning criteria and the logic of fuzzy concepts.
11500 *Journal of philosophical logic* 2(4), 458–508.
- 11501 Lample, G., M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer (2016). Neural
11502 architectures for named entity recognition. In *Proceedings of the North American Chapter*
11503 *of the Association for Computational Linguistics (NAACL)*, pp. 260–270.
- 11504 Langkilde, I. and K. Knight (1998). Generation that exploits corpus-based statistical
11505 knowledge. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 704–
11506 710.
- 11507 Lapata, M. (2003). Probabilistic text structuring: Experiments with sentence ordering. In
11508 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 545–552.
- 11509 Lappin, S. and H. J. Leass (1994). An algorithm for pronominal anaphora resolution.
11510 *Computational linguistics* 20(4), 535–561.
- 11511 Lari, K. and S. J. Young (1990). The estimation of stochastic context-free grammars using
11512 the inside-outside algorithm. *Computer speech & language* 4(1), 35–56.

- 11513 Lascarides, A. and N. Asher (2007). Segmented discourse representation theory: Dynamic
 11514 semantics with discourse structure. In *Computing meaning*, pp. 87–124. Springer.
- 11515 Law, E. and L. v. Ahn (2011). Human computation. *Synthesis Lectures on Artificial Intelli-*
 11516 *gence and Machine Learning* 5(3), 1–121.
- 11517 Lebret, R., D. Grangier, and M. Auli (2016). Neural text generation from structured data
 11518 with application to the biography domain. In *Proceedings of Empirical Methods for Natural*
 11519 *Language Processing (EMNLP)*, pp. 1203–1213.
- 11520 LeCun, Y. and Y. Bengio (1995). Convolutional networks for images, speech, and time
 11521 series. *The handbook of brain theory and neural networks* 3361.
- 11522 LeCun, Y., L. Bottou, G. B. Orr, and K.-R. Müller (1998). Efficient backprop. In *Neural*
 11523 *networks: Tricks of the trade*, pp. 9–50. Springer.
- 11524 Lee, C. M. and S. S. Narayanan (2005). Toward detecting emotions in spoken dialogs.
 11525 *IEEE transactions on speech and audio processing* 13(2), 293–303.
- 11526 Lee, H., A. Chang, Y. Peirsman, N. Chambers, M. Surdeanu, and D. Jurafsky (2013). De-
 11527 terministic coreference resolution based on entity-centric, precision-ranked rules. *Com-*
 11528 *putational Linguistics* 39(4), 885–916.
- 11529 Lee, H., Y. Peirsman, A. Chang, N. Chambers, M. Surdeanu, and D. Jurafsky (2011). Stan-
 11530 ford’s multi-pass sieve coreference resolution system at the conll-2011 shared task. In
 11531 *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 28–34. Associa-
 11532 tion for Computational Linguistics.
- 11533 Lee, K., L. He, M. Lewis, and L. Zettlemoyer (2017). End-to-end neural coreference reso-
 11534 lution. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11535 Lenat, D. B., R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd (1990). Cyc: toward
 11536 programs with common sense. *Communications of the ACM* 33(8), 30–49.
- 11537 Lesk, M. (1986). Automatic sense disambiguation using machine readable dictionaries:
 11538 how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual interna-*
 11539 *tional conference on Systems documentation*, pp. 24–26. ACM.
- 11540 Levesque, H. J., E. Davis, and L. Morgenstern (2011). The winograd schema challenge.
 11541 In *Aaai spring symposium: Logical formalizations of commonsense reasoning*, Volume 46, pp.
 11542 47.
- 11543 Levin, E., R. Pieraccini, and W. Eckert (1998). Using markov decision process for learning
 11544 dialogue strategies. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the*
 11545 *1998 IEEE International Conference on*, Volume 1, pp. 201–204. IEEE.

- 11546 Levy, O. and Y. Goldberg (2014). Dependency-based word embeddings. In *Proceedings of
11547 the Association for Computational Linguistics (ACL)*, pp. 302–308.
- 11548 Levy, O., Y. Goldberg, and I. Dagan (2015). Improving distributional similarity with
11549 lessons learned from word embeddings. *Transactions of the Association for Computational
11550 Linguistics* 3, 211–225.
- 11551 Lewis, M. and M. Steedman (2013). Combined distributional and logical semantics. *Trans-
11552 actions of the Association for Computational Linguistics* 1, 179–192.
- 11553 Lewis II, P. M. and R. E. Stearns (1968). Syntax-directed transduction. *Journal of the ACM
11554 (JACM)* 15(3), 465–488.
- 11555 Li, J. and D. Jurafsky (2015). Do multi-sense embeddings improve natural language
11556 understanding? In *Proceedings of Empirical Methods for Natural Language Processing
11557 (EMNLP)*, pp. 1722–1732.
- 11558 Li, J. and D. Jurafsky (2017). Neural net models of open-domain discourse coherence. In *Proceed-
11559 ings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 198–209.
- 11560 Li, J., R. Li, and E. Hovy (2014). Recursive deep models for discourse parsing. In *Proceed-
11561 ings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11562 Li, J., M.-T. Luong, and D. Jurafsky (2015). A hierarchical neural autoencoder for para-
11563 graphs and documents. In *Proceedings of Empirical Methods for Natural Language Process-
11564 ing (EMNLP)*.
- 11565 Li, J., T. Luong, D. Jurafsky, and E. Hovy (2015). When are tree structures necessary
11566 for deep learning of representations? In *Proceedings of Empirical Methods for Natural
11567 Language Processing (EMNLP)*, pp. 2304–2314.
- 11568 Li, J., W. Monroe, A. Ritter, D. Jurafsky, M. Galley, and J. Gao (2016, November). Deep
11569 reinforcement learning for dialogue generation. In *Proceedings of the 2016 Conference on
11570 Empirical Methods in Natural Language Processing*, Austin, Texas, pp. 1192–1202. Associa-
11571 tion for Computational Linguistics.
- 11572 Li, Q., S. Anzaroot, W.-P. Lin, X. Li, and H. Ji (2011). Joint inference for cross-document
11573 information extraction. In *Proceedings of the International Conference on Information and
11574 Knowledge Management (CIKM)*, pp. 2225–2228.
- 11575 Li, Q., H. Ji, and L. Huang (2013). Joint event extraction via structured prediction with
11576 global features. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
11577 73–82.

- 11578 Liang, P., A. Bouchard-Côté, D. Klein, and B. Taskar (2006). An end-to-end discriminative approach to machine translation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 761–768.
- 11581 Liang, P., M. Jordan, and D. Klein (2009). Learning semantic correspondences with less supervision. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 91–99.
- 11584 Liang, P., M. I. Jordan, and D. Klein (2013). Learning dependency-based compositional semantics. *Computational Linguistics* 39(2), 389–446.
- 11586 Liang, P. and D. Klein (2009). Online em for unsupervised models. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 611–619.
- 11589 Liang, P., S. Petrov, M. I. Jordan, and D. Klein (2007). The infinite pcfg using hierarchical dirichlet processes. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 688–697.
- 11592 Liang, P. and C. Potts (2015). Bringing machine learning and compositional semantics together. *Annual Review of Linguistics* 1(1), 355–376.
- 11594 Lieber, R. (2015). *Introducing morphology*. Cambridge University Press.
- 11595 Lin, D. (1998). Automatic retrieval and clustering of similar words. In *Proceedings of the 17th international conference on Computational linguistics-Volume 2*, pp. 768–774. Association for Computational Linguistics.
- 11598 Lin, J. and C. Dyer (2010). Data-intensive text processing with mapreduce. *Synthesis Lectures on Human Language Technologies* 3(1), 1–177.
- 11600 Lin, Z., M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio (2017). A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*.
- 11602 Lin, Z., M.-Y. Kan, and H. T. Ng (2009). Recognizing implicit discourse relations in the penn discourse treebank. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 343–351.
- 11605 Lin, Z., H. T. Ng, and M.-Y. Kan (2011). Automatically evaluating text coherence using discourse relations. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 997–1006.
- 11608 Lin, Z., H. T. Ng, and M. Y. Kan (2014, nov). A PDTB-styled end-to-end discourse parser. *Natural Language Engineering FirstView*, 1–34.

- 11610 Ling, W., C. Dyer, A. Black, and I. Trancoso (2015). Two/too simple adaptations of
11611 word2vec for syntax problems. In *Proceedings of the North American Chapter of the As-*
11612 *sociation for Computational Linguistics (NAACL)*.
- 11613 Ling, W., T. Luís, L. Marujo, R. F. Astudillo, S. Amir, C. Dyer, A. W. Black, and I. Trancoso
11614 (2015). Finding function in form: Compositional character models for open vocabulary
11615 word representation. In *Proceedings of Empirical Methods for Natural Language Processing*
11616 (*EMNLP*).
- 11617 Ling, W., G. Xiang, C. Dyer, A. Black, and I. Trancoso (2013). Microblogs as parallel cor-
11618 pora. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11619 Ling, X., S. Singh, and D. S. Weld (2015). Design challenges for entity linking. *Transactions*
11620 *of the Association for Computational Linguistics* 3, 315–328.
- 11621 Linguistic Data Consortium (2005, July). ACE (automatic content extraction) English an-
11622 notation guidelines for relations. Technical Report Version 5.8.3, Linguistic Data Con-
11623 sortium.
- 11624 Liu, B. (2015). *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. Cambridge
11625 University Press.
- 11626 Liu, D. C. and J. Nocedal (1989). On the limited memory BFGS method for large scale
11627 optimization. *Mathematical programming* 45(1-3), 503–528.
- 11628 Liu, Y., Q. Liu, and S. Lin (2006). Tree-to-string alignment template for statistical machine
11629 translation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 609–
11630 616.
- 11631 Loper, E. and S. Bird (2002). Nltk: The natural language toolkit. In *Proceedings of the ACL-*
11632 *02 Workshop on Effective tools and methodologies for teaching natural language processing and*
11633 *computational linguistics-Volume 1*, pp. 63–70. Association for Computational Linguistics.
- 11634 Louis, A., A. Joshi, and A. Nenkova (2010). Discourse indicators for content selection in
11635 summarization. In *Proceedings of the 11th Annual Meeting of the Special Interest Group on*
11636 *Discourse and Dialogue*, pp. 147–156. Association for Computational Linguistics.
- 11637 Louis, A. and A. Nenkova (2013). What makes writing great? first experiments on article
11638 quality prediction in the science journalism domain. *Transactions of the Association for*
11639 *Computational Linguistics* 1, 341–352.
- 11640 Loveland, D. W. (2016). *Automated Theorem Proving: a logical basis*. Elsevier.
- 11641 Lowe, R., N. Pow, I. V. Serban, and J. Pineau (2015). The ubuntu dialogue corpus: A large
11642 dataset for research in unstructured multi-turn dialogue systems. In *Proceedings of the*
11643 *Special Interest Group on Discourse and Dialogue (SIGDIAL)*.

- 11644 11645 Luo, X. (2005). On coreference resolution performance metrics. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 25–32.
- 11646 11647 11648 Luo, X., A. Ittycheriah, H. Jing, N. Kambhatla, and S. Roukos (2004). A mention-synchronous coreference resolution algorithm based on the bell tree. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11649 11650 Luong, M.-T., R. Socher, and C. D. Manning (2013). Better word representations with recursive neural networks for morphology. *CoNLL-2013* 104.
- 11651 11652 11653 Luong, T., H. Pham, and C. D. Manning (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1412–1421.
- 11654 11655 11656 Luong, T., I. Sutskever, Q. Le, O. Vinyals, and W. Zaremba (2015). Addressing the rare word problem in neural machine translation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 11–19.
- 11657 11658 11659 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)*.
- 11660 11661 Mairesse, F. and M. A. Walker (2011). Controlling user perceptions of linguistic style: Trainable generation of personality traits. *Computational Linguistics* 37(3), 455–488.
- 11662 11663 Malioutov, I. and R. Barzilay (2006). Minimum cut model for spoken lecture segmentation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 25–32.
- 11664 11665 11666 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.
- 11667 11668 Mann, W. C. and S. A. Thompson (1988). Rhetorical structure theory: Toward a functional theory of text organization. *Text* 8(3), 243–281.
- 11669 11670 Manning, C. D. (2015). Computational linguistics and deep learning. *Computational Linguistics* 41(4), 701–707.
- 11671 11672 Manning, C. D. (2016). Computational linguistics and deep learning. *Computational Linguistics* 41(4).
- 11673 11674 Manning, C. D., P. Raghavan, H. Schütze, et al. (2008). *Introduction to information retrieval*, Volume 1. Cambridge university press.
- 11675 11676 Manning, C. D. and H. Schütze (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts: MIT press.

- 11677 Marcu, D. (1996). Building up rhetorical structure trees. In *Proceedings of the National
11678 Conference on Artificial Intelligence*, pp. 1069–1074.
- 11679 Marcu, D. (1997a). From discourse structures to text summaries. In *Proceedings of the
11680 workshop on Intelligent Scalable Text Summarization*.
- 11681 Marcu, D. (1997b). From local to global coherence: A bottom-up approach to text plan-
11682 ning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 629–635.
- 11683 Marcus, M. P., M. A. Marcinkiewicz, and B. Santorini (1993). Building a large annotated
11684 corpus of English: The Penn Treebank. *Computational Linguistics* 19(2), 313–330.
- 11685 Maron, O. and T. Lozano-Pérez (1998). A framework for multiple-instance learning. In
11686 *Neural Information Processing Systems (NIPS)*, pp. 570–576.
- 11687 Márquez, G. G. (1970). *One Hundred Years of Solitude*. Harper & Row. English translation
11688 by Gregory Rabassa.
- 11689 Martins, A. F. T., N. A. Smith, and E. P. Xing (2009). Concise integer linear programming
11690 formulations for dependency parsing. In *Proceedings of the Association for Computational
11691 Linguistics (ACL)*, pp. 342–350.
- 11692 Martins, A. F. T., N. A. Smith, E. P. Xing, P. M. Q. Aguiar, and M. A. T. Figueiredo (2010).
11693 Turbo parsers: Dependency parsing by approximate variational inference. In *Proceed-
11694 ings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 34–44.
- 11695 Matsuzaki, T., Y. Miyao, and J. Tsujii (2005). Probabilistic cfg with latent annotations. In
11696 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 75–82.
- 11697 Matthiessen, C. and J. A. Bateman (1991). *Text generation and systemic-functional linguistics:
11698 experiences from English and Japanese*. Pinter Publishers.
- 11699 May, J. (2016). SemEval-2016 task 8: Meaning representation parsing. In *Proceedings of
11700 SemEval*, pp. 1063–1073.
- 11701 McCallum, A. and W. Li (2003). Early results for named entity recognition with condi-
11702 tional random fields, feature induction and web-enhanced lexicons. In *Proceedings of
11703 the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp.
11704 188–191.
- 11705 McCallum, A. and B. Wellner (2004). Conditional models of identity uncertainty with
11706 application to noun coreference. In *NIPS*, pp. 905–912.
- 11707 McDonald, R., K. Crammer, and F. Pereira (2005). Online large-margin training of depen-
11708 dency parsers. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
11709 91–98.

- 11710 McDonald, R., K. Hannan, T. Neylon, M. Wells, and J. Reynar (2007). Structured models
11711 for fine-to-coarse sentiment analysis. In *Proceedings of ACL*.
- 11712 McDonald, R. and F. Pereira (2006). Online learning of approximate dependency parsing
11713 algorithms. In *Proceedings of the European Chapter of the Association for Computational
11714 Linguistics (EACL)*.
- 11715 McDonald, R., F. Pereira, K. Ribarov, and J. Hajič (2005). Non-projective dependency
11716 parsing using spanning tree algorithms. In *Proceedings of Empirical Methods for Natural
11717 Language Processing (EMNLP)*, pp. 523–530.
- 11718 McKeown, K. (1992). *Text generation*. Cambridge University Press.
- 11719 McKeown, K., S. Rosenthal, K. Thadani, and C. Moore (2010). Time-efficient creation of
11720 an accurate sentence fusion corpus. In *Proceedings of the North American Chapter of the
11721 Association for Computational Linguistics (NAACL)*, pp. 317–320.
- 11722 McKeown, K. R., R. Barzilay, D. Evans, V. Hatzivassiloglou, J. L. Klavans, A. Nenkova,
11723 C. Sable, B. Schiffman, and S. Sigelman (2002). Tracking and summarizing news on a
11724 daily basis with columbia’s newsblaster. In *Proceedings of the second international confer-
11725 ence on Human Language Technology Research*, pp. 280–285.
- 11726 McNamee, P. and H. T. Dang (2009). Overview of the tac 2009 knowledge base population
11727 track. In *Text Analysis Conference (TAC)*, Volume 17, pp. 111–113.
- 11728 Medlock, B. and T. Briscoe (2007). Weakly supervised learning for hedge classification in
11729 scientific literature. In *Proceedings of the Association for Computational Linguistics (ACL)*,
11730 pp. 992–999.
- 11731 Mei, H., M. Bansal, and M. R. Walter (2016). What to talk about and how? selective gen-
11732 eration using lstms with coarse-to-fine alignment. In *Proceedings of the North American
11733 Chapter of the Association for Computational Linguistics (NAACL)*, pp. 720–730.
- 11734 Merity, S., N. S. Keskar, and R. Socher (2018). Regularizing and optimizing lstm language
11735 models. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 11736 Merity, S., C. Xiong, J. Bradbury, and R. Socher (2017). Pointer sentinel mixture models.
11737 In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 11738 Messud, C. (2014, June). A new ‘l’étranger’. *New York Review of Books*.
- 11739 Miao, Y. and P. Blunsom (2016). Language as a latent variable: Discrete generative mod-
11740 els for sentence compression. In *Proceedings of Empirical Methods for Natural Language
11741 Processing (EMNLP)*, pp. 319–328.

- 11742 Miao, Y., L. Yu, and P. Blunsom (2016). Neural variational inference for text processing. In
11743 *Proceedings of the International Conference on Machine Learning (ICML)*.
- 11744 Mihalcea, R., T. A. Chklovski, and A. Kilgarriff (2004, July). The senseval-3 english lexical
11745 sample task. In *Proceedings of SENSEVAL-3*, Barcelona, Spain, pp. 25–28. Association for
11746 Computational Linguistics.
- 11747 Mihalcea, R. and D. Radev (2011). *Graph-based natural language processing and information*
11748 *retrieval*. Cambridge University Press.
- 11749 Mikolov, T., K. Chen, G. Corrado, and J. Dean (2013). Efficient estimation of word repre-
11750 sentations in vector space. In *Proceedings of International Conference on Learning Represen-*
11751 *tations*.
- 11752 Mikolov, T., A. Deoras, D. Povey, L. Burget, and J. Cernocký (2011). Strategies for train-
11753 ing large scale neural network language models. In *Automatic Speech Recognition and*
11754 *Understanding (ASRU), 2011 IEEE Workshop on*, pp. 196–201. IEEE.
- 11755 Mikolov, T., M. Karafiat, L. Burget, J. Cernocký, and S. Khudanpur (2010). Recurrent
11756 neural network based language model. In *INTERSPEECH*, pp. 1045–1048.
- 11757 Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013). Distributed rep-
11758 resentations of words and phrases and their compositionality. In *Advances in Neural*
11759 *Information Processing Systems*, pp. 3111–3119.
- 11760 Mikolov, T., W.-t. Yih, and G. Zweig (2013). Linguistic regularities in continuous space
11761 word representations. In *Proceedings of the North American Chapter of the Association for*
11762 *Computational Linguistics (NAACL)*, pp. 746–751.
- 11763 Mikolov, T. and G. Zweig. Context dependent recurrent neural network language model.
11764 In *Proceedings of Spoken Language Technology (SLT)*, pp. 234–239.
- 11765 Miller, G. A., G. A. Heise, and W. Lichten (1951). The intelligibility of speech as a function
11766 of the context of the test materials. *Journal of experimental psychology* 41(5), 329.
- 11767 Miller, M., C. Sathi, D. Wiesenthal, J. Leskovec, and C. Potts (2011). Sentiment flow
11768 through hyperlink networks. In *Proceedings of the International Conference on Web and*
11769 *Social Media (ICWSM)*.
- 11770 Miller, S., J. Guinness, and A. Zamanian (2004). Name tagging with word clusters and
11771 discriminative training. In *Proceedings of the North American Chapter of the Association for*
11772 *Computational Linguistics (NAACL)*, pp. 337–342.
- 11773 Milne, D. and I. H. Witten (2008). Learning to link with wikipedia. In *Proceedings of the*
11774 *International Conference on Information and Knowledge Management (CIKM)*, pp. 509–518.
11775 ACM.

- 11776 Miltsakaki, E. and K. Kukich (2004). Evaluation of text coherence for electronic essay
 11777 scoring systems. *Natural Language Engineering* 10(1), 25–55.
- 11778 Minka, T. P. (1999). From hidden markov models to linear dynamical systems. Tech. Rep.
 11779 531, Vision and Modeling Group of Media Lab, MIT.
- 11780 Minsky, M. (1974). A framework for representing knowledge. Technical Report 306, MIT
 11781 AI Laboratory.
- 11782 Minsky, M. and S. Papert (1969). *Perceptrons*. MIT press.
- 11783 Mintz, M., S. Bills, R. Snow, and D. Jurafsky (2009). Distant supervision for relation extrac-
 11784 tion without labeled data. In *Proceedings of the Association for Computational Linguistics*
 11785 (ACL), pp. 1003–1011.
- 11786 Mirza, P., R. Sprugnoli, S. Tonelli, and M. Speranza (2014). Annotating causality in the
 11787 tempeval-3 corpus. In *Proceedings of the EACL 2014 Workshop on Computational Ap-
 11788 proaches to Causality in Language (CAtoCL)*, pp. 10–19.
- 11789 Misra, D. K. and Y. Artzi (2016). Neural shift-reduce ccg semantic parsing. In *Proceedings*
 11790 *of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11791 Mitchell, J. and M. Lapata (2010). Composition in distributional models of semantics.
 11792 *Cognitive Science* 34(8), 1388–1429.
- 11793 Miwa, M. and M. Bansal (2016). End-to-end relation extraction using lstms on sequences
 11794 and tree structures. In *Proceedings of the Association for Computational Linguistics (ACL)*,
 11795 pp. 1105–1116.
- 11796 Mnih, A. and G. Hinton (2007). Three new graphical models for statistical language mod-
 11797 ellng. In *Proceedings of the 24th international conference on Machine learning*, ICML '07,
 11798 New York, NY, USA, pp. 641–648. ACM.
- 11799 Mnih, A. and G. E. Hinton (2008). A scalable hierarchical distributed language model. In
 11800 *Neural Information Processing Systems (NIPS)*, pp. 1081–1088.
- 11801 Mnih, A. and Y. W. Teh (2012). A fast and simple algorithm for training neural probabilis-
 11802 tic language models. In *Proceedings of the International Conference on Machine Learning*
 11803 (ICML).
- 11804 Mohammad, S. M. and P. D. Turney (2013). Crowdsourcing a word–emotion association
 11805 lexicon. *Computational Intelligence* 29(3), 436–465.
- 11806 Mohri, M., F. Pereira, and M. Riley (2002). Weighted finite-state transducers in speech
 11807 recognition. *Computer Speech & Language* 16(1), 69–88.

- 11808 Mohri, M., A. Rostamizadeh, and A. Talwalkar (2012). *Foundations of machine learning*.
11809 MIT press.
- 11810 Montague, R. (1973). The proper treatment of quantification in ordinary english. In *Ap-*
11811 *proaches to natural language*, pp. 221–242. Springer.
- 11812 Moore, J. D. and C. L. Paris (1993, dec). Planning text for advisory dialogues: Capturing
11813 intentional and rhetorical information. *Comput. Linguist.* 19(4), 651–694.
- 11814 Morante, R. and E. Blanco (2012). *sem 2012 shared task: Resolving the scope and fo-
11815 cus of negation. In *Proceedings of the First Joint Conference on Lexical and Computational*
11816 *Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2:*
11817 *Proceedings of the Sixth International Workshop on Semantic Evaluation*, pp. 265–274. Asso-
11818 ciation for Computational Linguistics.
- 11819 Morante, R. and W. Daelemans (2009). Learning the scope of hedge cues in biomedical
11820 texts. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language*
11821 *Processing*, pp. 28–36. Association for Computational Linguistics.
- 11822 Morante, R. and C. Sporleder (2012). Modality and negation: An introduction to the
11823 special issue. *Computational linguistics* 38(2), 223–260.
- 11824 Mostafazadeh, N., A. Grelish, N. Chambers, J. Allen, and L. Vanderwende (2016, June).
11825 Caters: Causal and temporal relation scheme for semantic annotation of event struc-
11826 tures. In *Proceedings of the Fourth Workshop on Events*, San Diego, California, pp. 51–61.
11827 Association for Computational Linguistics.
- 11828 Mueller, T., H. Schmid, and H. Schütze (2013). Efficient higher-order CRFs for morpholog-
11829 ical tagging. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
11830 pp. 322–332.
- 11831 Müller, C. and M. Strube (2006). Multi-level annotation of linguistic data with mmax2.
11832 *Corpus technology and language pedagogy: New resources, new tools, new methods* 3, 197–
11833 214.
- 11834 Muralidharan, A. and M. A. Hearst (2013). Supporting exploratory text analysis in litera-
11835 ture study. *Literary and linguistic computing* 28(2), 283–295.
- 11836 Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.
- 11837 Nakagawa, T., K. Inui, and S. Kurohashi (2010). Dependency tree-based sentiment classi-
11838 fication using crfs with hidden variables. In *Proceedings of the North American Chapter of*
11839 *the Association for Computational Linguistics (NAACL)*, pp. 786–794.

- 11840 Nakazawa, T., M. Yaguchi, K. Uchimoto, M. Utiyama, E. Sumita, S. Kurohashi, and H. Isahara (2016). ASPEC: Asian scientific paper excerpt corpus. In *Proceedings of the Language Resources and Evaluation Conference*, pp. 2204–2208.
- 11843 Navigli, R. (2009). Word sense disambiguation: A survey. *ACM Computing Surveys (CSUR)* 41(2), 10.
- 11845 Neal, R. M. and G. E. Hinton (1998). A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pp. 355–368. Springer.
- 11847 Nenkova, A. and K. McKeown (2012). A survey of text summarization techniques. In *Mining text data*, pp. 43–76. Springer.
- 11849 Neubig, G. (2017). Neural machine translation and sequence-to-sequence models: A tutorial. *arXiv preprint arXiv:1703.01619*.
- 11851 Neubig, G., C. Dyer, Y. Goldberg, A. Matthews, W. Ammar, A. Anastasopoulos, M. Balles-teros, D. Chiang, D. Clothiaux, T. Cohn, K. Duh, M. Faruqui, C. Gan, D. Garrette, Y. Ji, L. Kong, A. Kuncoro, G. Kumar, C. Malaviya, P. Michel, Y. Oda, M. Richardson, N. Saphra, S. Swayamdipta, and P. Yin (2017). Dynet: The dynamic neural network toolkit.
- 11856 Neubig, G., Y. Goldberg, and C. Dyer (2017). On-the-fly operation batching in dynamic computation graphs. In *Neural Information Processing Systems (NIPS)*.
- 11858 Neuhaus, P. and N. Bröker (1997). The complexity of recognition of linguistically adequate dependency grammars. In *eacl*, pp. 337–343.
- 11860 Newman, D., C. Chemudugunta, and P. Smyth (2006). Statistical entity-topic models. In *Proceedings of Knowledge Discovery and Data Mining (KDD)*, pp. 680–686.
- 11862 Ng, V. (2010). Supervised noun phrase coreference research: The first fifteen years. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pp. 1396–1411. Association for Computational Linguistics.
- 11865 Nguyen, D. and A. S. Dogruöz (2013). Word level language identification in online multi-lingual communication. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11868 Nguyen, D. T. and S. Joty (2017). A neural local coherence model. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1320–1330.
- 11870 Nigam, K., A. K. McCallum, S. Thrun, and T. Mitchell (2000). Text classification from labeled and unlabeled documents using em. *Machine learning* 39(2-3), 103–134.

- 11872 Nirenburg, S. and Y. Wilks (2001). What's in a symbol: ontology, representation and lan-
11873 guage. *Journal of Experimental & Theoretical Artificial Intelligence* 13(1), 9–23.
- 11874 Nivre, J. (2008). Algorithms for deterministic incremental dependency parsing. *Computa-
11875 tional Linguistics* 34(4), 513–553.
- 11876 Nivre, J., M.-C. de Marneffe, F. Ginter, Y. Goldberg, J. Hajič, C. D. Manning, R. McDonald,
11877 S. Petrov, S. Pyysalo, N. Silveira, R. Tsarfaty, and D. Zeman (2016, may). Universal de-
11878 pendencies v1: A multilingual treebank collection. In N. C. C. Chair), K. Choukri, T. De-
11879 clerck, S. Goggi, M. Grobelnik, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk,
11880 and S. Piperidis (Eds.), *Proceedings of the Tenth International Conference on Language Re-
11881 sources and Evaluation (LREC 2016)*, Paris, France. European Language Resources Asso-
11882 ciation (ELRA).
- 11883 Nivre, J. and J. Nilsson (2005). Pseudo-projective dependency parsing. In *Proceedings of the
11884 43rd Annual Meeting on Association for Computational Linguistics*, pp. 99–106. Association
11885 for Computational Linguistics.
- 11886 Novikoff, A. B. (1962). On convergence proofs on perceptrons. In *Proceedings of the Sym-
11887 posium on the Mathematical Theory of Automata*, Volume 12, pp. 615—622.
- 11888 Och, F. J. and H. Ney (2003). A systematic comparison of various statistical alignment
11889 models. *Computational linguistics* 29(1), 19–51.
- 11890 O'Connor, B., M. Krieger, and D. Ahn (2010). Tweetmotif: Exploratory search and topic
11891 summarization for twitter. In *Proceedings of the International Conference on Web and Social
11892 Media (ICWSM)*, pp. 384–385.
- 11893 Oflazer, K. and İ. Kuruöz (1994). Tagging and morphological disambiguation of turkish
11894 text. In *Proceedings of the fourth conference on Applied natural language processing*, pp. 144–
11895 149. Association for Computational Linguistics.
- 11896 Ohta, T., Y. Tateisi, and J.-D. Kim (2002). The genia corpus: An annotated research abstract
11897 corpus in molecular biology domain. In *Proceedings of the second international conference
11898 on Human Language Technology Research*, pp. 82–86. Morgan Kaufmann Publishers Inc.
- 11899 Onishi, T., H. Wang, M. Bansal, K. Gimpel, and D. McAllester (2016). Who did what: A
11900 large-scale person-centered cloze dataset. In *Proceedings of Empirical Methods for Natural
11901 Language Processing (EMNLP)*, pp. 2230–2235.
- 11902 Owoputi, O., B. O'Connor, C. Dyer, K. Gimpel, N. Schneider, and N. A. Smith (2013).
11903 Improved part-of-speech tagging for online conversational text with word clusters. In
11904 *Proceedings of the North American Chapter of the Association for Computational Linguistics
11905 (NAACL)*, pp. 380–390.

- 11906 Packard, W., E. M. Bender, J. Read, S. Oepen, and R. Dridan (2014). Simple negation
 11907 scope resolution through deep parsing: A semantic solution to a semantic problem. In
 11908 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 69–78.
- 11909 Paice, C. D. (1990). Another stemmer. In *ACM SIGIR Forum*, Volume 24, pp. 56–61.
- 11910 Pak, A. and P. Paroubek (2010). Twitter as a corpus for sentiment analysis and opinion
 11911 mining. In *LREC*, Volume 10, pp. 1320–1326.
- 11912 Palmer, F. R. (2001). *Mood and modality*. Cambridge University Press.
- 11913 Palmer, M., D. Gildea, and P. Kingsbury (2005). The proposition bank: An annotated
 11914 corpus of semantic roles. *Computational linguistics* 31(1), 71–106.
- 11915 Pan, S. J. and Q. Yang (2010). A survey on transfer learning. *IEEE Transactions on knowledge
 11916 and data engineering* 22(10), 1345–1359.
- 11917 Pan, X., T. Cassidy, U. Hermjakob, H. Ji, and K. Knight (2015). Unsupervised entity linking
 11918 with abstract meaning representation. In *Proceedings of the North American Chapter of the
 11919 Association for Computational Linguistics (NAACL)*, pp. 1130–1139.
- 11920 Pang, B. and L. Lee (2004). A sentimental education: Sentiment analysis using subjectivity
 11921 summarization based on minimum cuts. In *Proceedings of the Association for Compu-
 11922 tational Linguistics (ACL)*, pp. 271–278.
- 11923 Pang, B. and L. Lee (2005). Seeing stars: Exploiting class relationships for sentiment cate-
 11924 gorization with respect to rating scales. In *Proceedings of the Association for Computational
 11925 Linguistics (ACL)*, pp. 115–124.
- 11926 Pang, B. and L. Lee (2008). Opinion mining and sentiment analysis. *Foundations and trends
 11927 in information retrieval* 2(1–2), 1–135.
- 11928 Pang, B., L. Lee, and S. Vaithyanathan (2002). Thumbs up?: sentiment classification using
 11929 machine learning techniques. In *Proceedings of Empirical Methods for Natural Language
 11930 Processing (EMNLP)*, pp. 79–86.
- 11931 Papineni, K., S. Roukos, T. Ward, and W.-J. Zhu (2002). Bleu: a method for automatic
 11932 evaluation of machine translation. In *Proceedings of the Association for Computational
 11933 Linguistics (ACL)*, pp. 311–318.
- 11934 Park, J. and C. Cardie (2012). Improving implicit discourse relation recognition through
 11935 feature set optimization. In *Proceedings of the Special Interest Group on Discourse and Dia-
 11936 logue (SIGDIAL)*, pp. 108–112.
- 11937 Parsons, T. (1990). *Events in the Semantics of English*, Volume 5. MIT Press.

- 11938 Pascanu, R., T. Mikolov, and Y. Bengio (2013). On the difficulty of training recurrent neural
11939 networks. In *Proceedings of the 30th International Conference on Machine Learning (ICML-*
11940 *13)*, pp. 1310–1318.
- 11941 Paul, M., M. Federico, and S. Stüker (2010). Overview of the iwslt 2010 evaluation cam-
11942 paign. In *International Workshop on Spoken Language Translation (IWSLT) 2010*.
- 11943 Pedersen, T., S. Patwardhan, and J. Michelizzi (2004). Wordnet::similarity - measuring the
11944 relatedness of concepts. In *Proceedings of the North American Chapter of the Association for*
11945 *Computational Linguistics (NAACL)*, pp. 38–41.
- 11946 Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blon-
11947 del, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau,
11948 M. Brucher, M. Perrot, and E. Duchesnay (2011). Scikit-learn: Machine learning in
11949 Python. *Journal of Machine Learning Research* 12, 2825–2830.
- 11950 Pei, W., T. Ge, and B. Chang (2015). An effective neural network model for graph-based
11951 dependency parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*,
11952 pp. 313–322.
- 11953 Peldszus, A. and M. Stede (2013). From argument diagrams to argumentation mining
11954 in texts: A survey. *International Journal of Cognitive Informatics and Natural Intelligence*
11955 (*IJCINI*) 7(1), 1–31.
- 11956 Peldszus, A. and M. Stede (2015). An annotated corpus of argumentative microtexts. In
11957 *Proceedings of the First Conference on Argumentation*.
- 11958 Peng, F., F. Feng, and A. McCallum (2004). Chinese segmentation and new word detec-
11959 tion using conditional random fields. In *Proceedings of the International Conference on*
11960 *Computational Linguistics (COLING)*, pp. 562.
- 11961 Pennington, J., R. Socher, and C. Manning (2014). Glove: Global vectors for word repre-
11962 sentation. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
11963 pp. 1532–1543.
- 11964 Pereira, F. and Y. Schabes (1992). Inside-outside reestimation from partially bracketed
11965 corpora. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 128–
11966 135.
- 11967 Pereira, F. C. N. and S. M. Shieber (2002). *Prolog and natural-language analysis*. Microtome
11968 Publishing.
- 11969 Peterson, W. W., T. G. Birdsall, and W. C. Fox (1954). The theory of signal detectability.
11970 *Transactions of the IRE professional group on information theory* 4(4), 171–212.

- 11971 Petrov, S., L. Barrett, R. Thibaux, and D. Klein (2006). Learning accurate, compact, and in-
 11972 terpretable tree annotation. In *Proceedings of the Association for Computational Linguistics*
 11973 (*ACL*).
- 11974 Petrov, S., D. Das, and R. McDonald (2012, May). A universal part-of-speech tagset. In
 11975 *Proceedings of LREC*.
- 11976 Petrov, S. and R. McDonald (2012). Overview of the 2012 shared task on parsing the web.
 11977 In *Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL)*,
 11978 Volume 59.
- 11979 Pinker, S. (2003). *The language instinct: How the mind creates language*. Penguin UK.
- 11980 Pinter, Y., R. Guthrie, and J. Eisenstein (2017). Mimicking word embeddings using
 11981 subword RNNs. In *Proceedings of Empirical Methods for Natural Language Processing*
 11982 (*EMNLP*).
- 11983 Pitler, E., A. Louis, and A. Nenkova (2009). Automatic sense prediction for implicit dis-
 11984 course relations in text. In *Proceedings of the Association for Computational Linguistics*
 11985 (*ACL*).
- 11986 Pitler, E. and A. Nenkova (2009). Using syntax to disambiguate explicit discourse con-
 11987 nectives in text. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
 11988 13–16.
- 11989 Pitler, E., M. Raghupathy, H. Mehta, A. Nenkova, A. Lee, and A. Joshi (2008). Easily iden-
 11990 tifiable discourse relations. In *Proceedings of the International Conference on Computational*
 11991 *Linguistics (COLING)*, pp. 87–90.
- 11992 Plank, B., A. Søgaard, and Y. Goldberg (2016). Multilingual part-of-speech tagging with
 11993 bidirectional long short-term memory models and auxiliary loss. In *Proceedings of the*
 11994 *Association for Computational Linguistics (ACL)*.
- 11995 Poesio, M., R. Stevenson, B. Di Eugenio, and J. Hitzeman (2004). Centering: A parametric
 11996 theory and its instantiations. *Computational linguistics* 30(3), 309–363.
- 11997 Polanyi, L. and A. Zaenen (2006). Contextual valence shifters. In *Computing attitude and*
 11998 *affect in text: Theory and applications*. Springer.
- 11999 Ponzetto, S. P. and M. Strube (2006). Exploiting semantic role labeling, wordnet and
 12000 wikipedia for coreference resolution. In *Proceedings of the North American Chapter of*
 12001 *the Association for Computational Linguistics (NAACL)*, pp. 192–199.
- 12002 Ponzetto, S. P. and M. Strube (2007). Knowledge derived from wikipedia for computing
 12003 semantic relatedness. *Journal of Artificial Intelligence Research* 30, 181–212.

- 12004 Poon, H. and P. Domingos (2008). Joint unsupervised coreference resolution with markov
12005 logic. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.
12006 650–659.
- 12007 Poon, H. and P. Domingos (2009). Unsupervised semantic parsing. In *Proceedings of Em-
12008 pirical Methods for Natural Language Processing (EMNLP)*, pp. 1–10.
- 12009 Popel, M., D. Marecek, J. Stepánek, D. Zeman, and Z. Zabokrtský (2013). Coordination
12010 structures in dependency treebanks. In *Proceedings of the Association for Computational
12011 Linguistics (ACL)*, pp. 517–527.
- 12012 Popescu, A.-M., O. Etzioni, and H. Kautz (2003). Towards a theory of natural language
12013 interfaces to databases. In *Proceedings of Intelligent User Interfaces (IUI)*, pp. 149–157.
- 12014 Poplack, S. (1980). Sometimes i'll start a sentence in spanish y termino en español: toward
12015 a typology of code-switching1. *Linguistics* 18(7-8), 581–618.
- 12016 Porter, M. F. (1980). An algorithm for suffix stripping. *Program* 14(3), 130–137.
- 12017 Prabhakaran, V., O. Rambow, and M. Diab (2010). Automatic committed belief tagging.
12018 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.
12019 1014–1022.
- 12020 Pradhan, S., X. Luo, M. Recasens, E. Hovy, V. Ng, and M. Strube (2014). Scoring corefer-
12021 ence partitions of predicted mentions: A reference implementation. In *Proceedings of the
12022 Association for Computational Linguistics (ACL)*, pp. 30–35.
- 12023 Pradhan, S., L. Ramshaw, M. Marcus, M. Palmer, R. Weischedel, and N. Xue (2011).
12024 CoNLL-2011 shared task: Modeling unrestricted coreference in OntoNotes. In *Proceed-
12025 ings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*,
12026 pp. 1–27. Association for Computational Linguistics.
- 12027 Pradhan, S., W. Ward, K. Hacioglu, J. H. Martin, and D. Jurafsky (2005). Semantic role
12028 labeling using different syntactic views. In *Proceedings of the Association for Computational
12029 Linguistics (ACL)*, pp. 581–588.
- 12030 Prasad, R., N. Dinesh, A. Lee, E. Miltsakaki, L. Robaldo, A. Joshi, and B. Webber (2008).
12031 The Penn Discourse Treebank 2.0. In *Proceedings of LREC*.
- 12032 Punyakanok, V., D. Roth, and W.-t. Yih (2008). The importance of syntactic parsing and
12033 inference in semantic role labeling. *Computational Linguistics* 34(2), 257–287.
- 12034 Pustejovsky, J., P. Hanks, R. Sauri, A. See, R. Gaizauskas, A. Setzer, D. Radev, B. Sundheim,
12035 D. Day, L. Ferro, et al. (2003). The timebank corpus. In *Corpus linguistics*, Volume 2003,
12036 pp. 40. Lancaster, UK.

- 12037 Pustejovsky, J., B. Ingria, R. Sauri, J. Castano, J. Littman, R. Gaizauskas, A. Setzer, G. Katz,
 12038 and I. Mani (2005). The specification language timeml. In *The language of time: A reader*,
 12039 pp. 545–557. Oxford University Press.
- 12040 Qin, L., Z. Zhang, H. Zhao, Z. Hu, and E. Xing (2017). Adversarial connective-exploiting
 12041 networks for implicit discourse relation classification. In *Proceedings of the Association*
 12042 for *Computational Linguistics (ACL)*, pp. 1006–1017.
- 12043 Qiu, G., B. Liu, J. Bu, and C. Chen (2011). Opinion word expansion and target extraction
 12044 through double propagation. *Computational linguistics* 37(1), 9–27.
- 12045 Quattoni, A., S. Wang, L.-P. Morency, M. Collins, and T. Darrell (2007). Hidden conditional
 12046 random fields. *IEEE transactions on pattern analysis and machine intelligence* 29(10).
- 12047 Rahman, A. and V. Ng (2011). Narrowing the modeling gap: a cluster-ranking approach
 12048 to coreference resolution. *Journal of Artificial Intelligence Research* 40, 469–521.
- 12049 Rajpurkar, P., J. Zhang, K. Lopyrev, and P. Liang (2016). Squad: 100,000+ questions for
 12050 machine comprehension of text. In *Proceedings of Empirical Methods for Natural Language*
 12051 *Processing (EMNLP)*, pp. 2383–2392.
- 12052 Ranzato, M., S. Chopra, M. Auli, and W. Zaremba (2016). Sequence level training with
 12053 recurrent neural networks. In *Proceedings of the International Conference on Learning Rep-*
 12054 *resentations (ICLR)*.
- 12055 Rao, D., D. Yarowsky, A. Shreevats, and M. Gupta (2010). Classifying latent user attributes
 12056 in twitter. In *Proceedings of Workshop on Search and mining user-generated contents*.
- 12057 Ratinov, L. and D. Roth (2009). Design challenges and misconceptions in named entity
 12058 recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language*
 12059 *Learning*, pp. 147–155. Association for Computational Linguistics.
- 12060 Ratinov, L., D. Roth, D. Downey, and M. Anderson (2011). Local and global algorithms
 12061 for disambiguation to wikipedia. In *Proceedings of the Association for Computational Lin-*
 12062 *guistics (ACL)*, pp. 1375–1384.
- 12063 Ratliff, N. D., J. A. Bagnell, and M. Zinkevich (2007). (approximate) subgradient methods
 12064 for structured prediction. In *Proceedings of Artificial Intelligence and Statistics (AISTATS)*,
 12065 pp. 380–387.
- 12066 Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In *emnlp*,
 12067 pp. 133–142.
- 12068 Ratnaparkhi, A., J. Reynar, and S. Roukos (1994). A maximum entropy model for preposi-
 12069 tional phrase attachment. In *Proceedings of the workshop on Human Language Technology*,
 12070 pp. 250–255. Association for Computational Linguistics.

- 12071 Read, J. (2005). Using emoticons to reduce dependency in machine learning techniques for
12072 sentiment classification. In *Proceedings of the ACL student research workshop*, pp. 43–48.
12073 Association for Computational Linguistics.
- 12074 Reisinger, D., R. Rudinger, F. Ferraro, C. Harman, K. Rawlins, and B. V. Durme (2015).
12075 Semantic proto-roles. *Transactions of the Association for Computational Linguistics* 3, 475–
12076 488.
- 12077 Reisinger, J. and R. J. Mooney (2010). Multi-prototype vector-space models of word mean-
12078 ing. In *Proceedings of the North American Chapter of the Association for Computational Lin-*
12079 *guistics (NAACL)*, pp. 109–117.
- 12080 Reiter, E. and R. Dale (2000). *Building natural language generation systems*. Cambridge
12081 university press.
- 12082 Resnik, P., M. B. Olsen, and M. Diab (1999). The bible as a parallel corpus: Annotating the
12083 ‘book of 2000 tongues’. *Computers and the Humanities* 33(1-2), 129–153.
- 12084 Resnik, P. and N. A. Smith (2003). The web as a parallel corpus. *Computational Linguis-*
12085 *tics* 29(3), 349–380.
- 12086 Ribeiro, F. N., M. Araújo, P. Gonçalves, M. A. Gonçalves, and F. Benevenuto (2016).
12087 Sentibench-a benchmark comparison of state-of-the-practice sentiment analysis meth-
12088 ods. *EPJ Data Science* 5(1), 1–29.
- 12089 Richardson, M., C. J. Burges, and E. Renshaw (2013). MCTest: A challenge dataset for
12090 the open-domain machine comprehension of text. In *Proceedings of Empirical Methods for*
12091 *Natural Language Processing (EMNLP)*, pp. 193–203.
- 12092 Riedel, S., L. Yao, and A. McCallum (2010). Modeling relations and their mentions without
12093 labeled text. In *Proceedings of the European Conference on Machine Learning and Principles*
12094 *and Practice of Knowledge Discovery in Databases (ECML)*, pp. 148–163.
- 12095 Riedl, M. O. and R. M. Young (2010). Narrative planning: Balancing plot and character.
12096 *Journal of Artificial Intelligence Research* 39, 217–268.
- 12097 Rieser, V. and O. Lemon (2011). *Reinforcement learning for adaptive dialogue systems: a data-
12098 driven methodology for dialogue management and natural language generation*. Springer Sci-
12099 ence & Business Media.
- 12100 Riloff, E. (1996). Automatically generating extraction patterns from untagged text. In
12101 *Proceedings of the national conference on artificial intelligence*, pp. 1044–1049.
- 12102 Riloff, E. and J. Wiebe (2003). Learning extraction patterns for subjective expressions. In
12103 *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pp.
12104 105–112. Association for Computational Linguistics.

- 12105 Ritchie, G. (2001). Current directions in computational humour. *Artificial Intelligence Review* 16(2), 119–135.
- 12107 Ritter, A., C. Cherry, and W. B. Dolan (2011). Data-driven response generation in social
12108 media. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.
12109 583–593.
- 12110 Roark, B., M. Saraclar, and M. Collins (2007). Discriminative n_1/n_2 -gram language
12111 modeling. *Computer Speech & Language* 21(2), 373–392.
- 12112 Robert, C. and G. Casella (2013). *Monte Carlo statistical methods*. Springer Science & Busi-
12113 ness Media.
- 12114 Rosenfeld, R. (1996). A maximum entropy approach to adaptive statistical language mod-
12115 elling. *Computer Speech & Language* 10(3), 187–228.
- 12116 Ross, S., G. Gordon, and D. Bagnell (2011). A reduction of imitation learning and struc-
12117 tured prediction to no-regret online learning. In *Proceedings of Artificial Intelligence and*
12118 *Statistics (AISTATS)*, pp. 627–635.
- 12119 Roy, N., J. Pineau, and S. Thrun (2000). Spoken dialogue management using probabilistic
12120 reasoning. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 93–
12121 100.
- 12122 Rudnicky, A. and W. Xu (1999). An agenda-based dialog management architecture for
12123 spoken language systems. In *IEEE Automatic Speech Recognition and Understanding Work-
shop*, Volume 13.
- 12125 Rush, A. M., S. Chopra, and J. Weston (2015). A neural attention model for abstractive sen-
12126 tence summarization. In *Proceedings of Empirical Methods for Natural Language Processing*
12127 (*EMNLP*), pp. 379–389.
- 12128 Rush, A. M., D. Sontag, M. Collins, and T. Jaakkola (2010). On dual decomposition and
12129 linear programming relaxations for natural language processing. In *Proceedings of Em-
12130 pirical Methods for Natural Language Processing (EMNLP)*, pp. 1–11.
- 12131 Russell, S. J. and P. Norvig (2009). *Artificial intelligence: a modern approach* (3rd ed.). Prentice
12132 Hall.
- 12133 Rutherford, A., V. Demberg, and N. Xue (2017). A systematic study of neural discourse
12134 models for implicit discourse relation. In *Proceedings of the European Chapter of the Asso-
12135 ciation for Computational Linguistics (EACL)*, pp. 281–291.
- 12136 Rutherford, A. T. and N. Xue (2014). Discovering implicit discourse relations through
12137 brown cluster pair representation and coreference patterns. In *Proceedings of the Euro-
12138 pean Chapter of the Association for Computational Linguistics (EACL)*.

- 12139 Sag, I. A., T. Baldwin, F. Bond, A. Copestake, and D. Flickinger (2002). Multiword expressions: A pain in the neck for nlp. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pp. 1–15. Springer.
- 12142 Sagae, K. Analysis of discourse structure with syntactic dependencies and data-driven shift-reduce parsing. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, Paris, France, pp. 81–84. Association for Computational Linguistics.
- 12145 Santos, C. D. and B. Zadrozny (2014). Learning character-level representations for part-of-speech tagging. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1818–1826.
- 12148 Sato, M.-A. and S. Ishii (2000). On-line em algorithm for the normalized gaussian network. *Neural computation* 12(2), 407–432.
- 12150 Saurí, R. and J. Pustejovsky (2009). Factbank: a corpus annotated with event factuality. *Language resources and evaluation* 43(3), 227.
- 12152 Saxe, A. M., J. L. McClelland, and S. Ganguli (2014). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 12155 Schank, R. C. and R. Abelson (1977). *Scripts, goals, plans, and understanding*. Hillsdale, NJ: Erlbaum.
- 12157 Schapire, R. E. and Y. Singer (2000). Boostexter: A boosting-based system for text categorization. *Machine learning* 39(2-3), 135–168.
- 12159 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.
- 12161 Schnabel, T., I. Labutov, D. Mimno, and T. Joachims (2015). Evaluation methods for unsupervised word embeddings. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 298–307.
- 12164 Schneider, N., J. Flanigan, and T. O’Gorman (2015). The logic of amr: Practical, unified, graph-based sentence semantics for nlp. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 4–5.
- 12167 Schütze, H. (1998). Automatic word sense discrimination. *Computational linguistics* 24(1), 97–123.
- 12169 Schwarm, S. E. and M. Ostendorf (2005). Reading level assessment using support vector machines and statistical language models. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 523–530.

- 12172 See, A., P. J. Liu, and C. D. Manning (2017). Get to the point: Summarization with pointer-
 12173 generator networks. In *Proceedings of the Association for Computational Linguistics (ACL)*,
 12174 pp. 1073–1083.
- 12175 Sennrich, R., B. Haddow, and A. Birch (2016). Neural machine translation of rare words
 12176 with subword units. In *Proceedings of the Association for Computational Linguistics (ACL)*,
 12177 pp. 1715–1725.
- 12178 Serban, I. V., A. Sordoni, Y. Bengio, A. C. Courville, and J. Pineau (2016). Building end-to-
 12179 end dialogue systems using generative hierarchical neural network models. In *Proceed-
 12180 ings of the National Conference on Artificial Intelligence (AAAI)*, pp. 3776–3784.
- 12181 Settles, B. (2012). Active learning. *Synthesis Lectures on Artificial Intelligence and Machine
 12182 Learning* 6(1), 1–114.
- 12183 Shang, L., Z. Lu, and H. Li (2015). Neural responding machine for short-text conversation.
 12184 In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1577–1586.
- 12185 Shen, D. and M. Lapata (2007). Using semantic roles to improve question answering. In
 12186 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 12–21.
- 12187 Shen, S., Y. Cheng, Z. He, W. He, H. Wu, M. Sun, and Y. Liu (2016). Minimum risk train-
 12188 ing for neural machine translation. In *Proceedings of the Association for Computational
 12189 Linguistics (ACL)*, pp. 1683–1692.
- 12190 Shen, W., J. Wang, and J. Han (2015). Entity linking with a knowledge base: Issues, tech-
 12191 niques, and solutions. *IEEE Transactions on Knowledge and Data Engineering* 27(2), 443–
 12192 460.
- 12193 Shieber, S. M. (1985). Evidence against the context-freeness of natural language. *Linguistics
 12194 and Philosophy* 8(3), 333–343.
- 12195 Siegelmann, H. T. and E. D. Sontag (1995). On the computational power of neural nets.
 12196 *Journal of computer and system sciences* 50(1), 132–150.
- 12197 Singh, S., A. Subramanya, F. Pereira, and A. McCallum (2011). Large-scale cross-
 12198 document coreference using distributed inference and hierarchical models. In *Proceed-
 12199 ings of the Association for Computational Linguistics (ACL)*, pp. 793–803.
- 12200 Sipser, M. (2012). *Introduction to the Theory of Computation*. Cengage Learning.
- 12201 Smith, D. A. and J. Eisner (2006). Minimum risk annealing for training log-linear models.
 12202 In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 787–794.
- 12203 Smith, D. A. and J. Eisner (2008). Dependency parsing by belief propagation. In *Proceed-
 12204 ings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 145–156.

- 12205 Smith, D. A. and N. A. Smith (2007). Probabilistic models of nonprojective dependency
12206 trees. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.
12207 132–140.
- 12208 Smith, N. A. (2011). Linguistic structure prediction. *Synthesis Lectures on Human Language
12209 Technologies* 4(2), 1–274.
- 12210 Snover, M., B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul (2006). A study of transla-
12211 tion edit rate with targeted human annotation. In *Proceedings of association for machine
12212 translation in the Americas*, Volume 200.
- 12213 Snow, R., B. O'Connor, D. Jurafsky, and A. Y. Ng (2008). Cheap and fast—but is it good?:
12214 evaluating non-expert annotations for natural language tasks. In *Proceedings of Empirical
12215 Methods for Natural Language Processing (EMNLP)*, pp. 254–263.
- 12216 Snyder, B. and R. Barzilay (2007). Database-text alignment via structured multilabel classi-
12217 fication. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*,
12218 pp. 1713–1718.
- 12219 Socher, R., J. Bauer, C. D. Manning, and A. Y. Ng (2013). Parsing with compositional vector
12220 grammars. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 12221 Socher, R., B. Huval, C. D. Manning, and A. Y. Ng (2012). Semantic compositionality
12222 through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Em-
12223 pirical Methods in Natural Language Processing and Computational Natural Language Learn-
12224 ing*, pp. 1201–1211. Association for Computational Linguistics.
- 12225 Socher, R., A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts (2013).
12226 Recursive deep models for semantic compositionality over a sentiment treebank. In
12227 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12228 Søgaard, A. (2013). Semi-supervised learning and domain adaptation in natural language
12229 processing. *Synthesis Lectures on Human Language Technologies* 6(2), 1–103.
- 12230 Solorio, T. and Y. Liu (2008). Learning to predict code-switching points. In *Proceedings
12231 of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 973–981. Association
12232 for Computational Linguistics.
- 12233 Somasundaran, S., G. Namata, J. Wiebe, and L. Getoor (2009). Supervised and unsuper-
12234 vised methods in employing discourse relations for improving opinion polarity classi-
12235 fication. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12236 Somasundaran, S. and J. Wiebe (2009). Recognizing stances in online debates. In *Proceed-
12237 ings of the Association for Computational Linguistics (ACL)*, pp. 226–234.

- 12238 Song, L., B. Boots, S. M. Siddiqi, G. J. Gordon, and A. J. Smola (2010). Hilbert space
 12239 embeddings of hidden markov models. In *Proceedings of the International Conference on*
 12240 *Machine Learning (ICML)*, pp. 991–998.
- 12241 Song, L., Y. Zhang, X. Peng, Z. Wang, and D. Gildea (2016). Amr-to-text generation as
 12242 a traveling salesman problem. In *Proceedings of Empirical Methods for Natural Language*
 12243 *Processing (EMNLP)*, pp. 2084–2089.
- 12244 Soon, W. M., H. T. Ng, and D. C. Y. Lim (2001). A machine learning approach to corefer-
 12245 ence resolution of noun phrases. *Computational linguistics* 27(4), 521–544.
- 12246 Sordoni, A., M. Galley, M. Auli, C. Brockett, Y. Ji, M. Mitchell, J.-Y. Nie, J. Gao, and B. Dolan
 12247 (2015). A neural network approach to context-sensitive generation of conversational
 12248 responses. In *Proceedings of the North American Chapter of the Association for Computational*
 12249 *Linguistics (NAACL)*.
- 12250 Soriceut, R. and D. Marcu (2003). Sentence level discourse parsing using syntactic and
 12251 lexical information. In *Proceedings of the North American Chapter of the Association for*
 12252 *Computational Linguistics (NAACL)*, pp. 149–156.
- 12253 Sowa, J. F. (2000). *Knowledge representation: logical, philosophical, and computational founda-
 12254 tions*. Pacific Grove, CA: Brooks/Cole.
- 12255 Spärck Jones, K. (1972). A statistical interpretation of term specificity and its application
 12256 in retrieval. *Journal of documentation* 28(1), 11–21.
- 12257 Spitkovsky, V. I., H. Alshawi, D. Jurafsky, and C. D. Manning (2010). Viterbi training
 12258 improves unsupervised dependency parsing. In *CONLL*, pp. 9–17.
- 12259 Sporleder, C. and M. Lapata (2005). Discourse chunking and its application to sen-
 12260 tence compression. In *Proceedings of Empirical Methods for Natural Language Processing*
 12261 (*EMNLP*), pp. 257–264.
- 12262 Sproat, R., A. Black, S. Chen, S. Kumar, M. Ostendorf, and C. Richards (2001). Normaliza-
 12263 tion of non-standard words. *Computer Speech & Language* 15(3), 287–333.
- 12264 Sproat, R., W. Gale, C. Shih, and N. Chang (1996). A stochastic finite-state word-
 12265 segmentation algorithm for chinese. *Computational linguistics* 22(3), 377–404.
- 12266 Sra, S., S. Nowozin, and S. J. Wright (2012). *Optimization for machine learning*. MIT Press.
- 12267 Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014).
 12268 Dropout: A simple way to prevent neural networks from overfitting. *The Journal of*
 12269 *Machine Learning Research* 15(1), 1929–1958.

- 12270 Srivastava, R. K., K. Greff, and J. Schmidhuber (2015). Training very deep networks. In
12271 *Neural Information Processing Systems (NIPS)*, pp. 2377–2385.
- 12272 Stab, C. and I. Gurevych (2014a). Annotating argument components and relations in per-
12273 suasive essays. In *Proceedings of the International Conference on Computational Linguistics*
12274 (*COLING*), pp. 1501–1510.
- 12275 Stab, C. and I. Gurevych (2014b). Identifying argumentative discourse structures in per-
12276 suasive essays. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Lan-*
12277 *guage Processing (EMNLP)*, pp. 46–56.
- 12278 Stede, M. (2011, nov). *Discourse Processing*, Volume 4 of *Synthesis Lectures on Human Lan-*
12279 *guage Technologies*. Morgan & Claypool Publishers.
- 12280 Steedman, M. and J. Baldridge (2011). Combinatory categorial grammar. In *Non-*
12281 *Transformational Syntax: Formal and Explicit Models of Grammar*. Wiley-Blackwell.
- 12282 Stenetorp, P., S. Pyysalo, G. Topić, T. Ohta, S. Ananiadou, and J. Tsujii (2012). Brat: a web-
12283 based tool for nlp-assisted text annotation. In *Proceedings of the European Chapter of the*
12284 *Association for Computational Linguistics (EACL)*, pp. 102–107.
- 12285 Stern, M., J. Andreas, and D. Klein (2017). A minimal span-based neural constituency
12286 parser. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 12287 Stolcke, A., K. Ries, N. Coccaro, E. Shriberg, R. Bates, D. Jurafsky, P. Taylor, R. Martin,
12288 C. Van Ess-Dykema, and M. Meteer (2000). Dialogue act modeling for automatic tag-
12289 ging and recognition of conversational speech. *Computational linguistics* 26(3), 339–373.
- 12290 Stone, P. J. (1966). *The General Inquirer: A Computer Approach to Content Analysis*. The MIT
12291 Press.
- 12292 Stoyanov, V., N. Gilbert, C. Cardie, and E. Riloff (2009). Conundrums in noun phrase
12293 coreference resolution: Making sense of the state-of-the-art. In *Proceedings of the Associa-*
12294 *tion for Computational Linguistics (ACL)*, pp. 656–664.
- 12295 Strang, G. (2016). *Introduction to linear algebra* (Fifth ed.). Wellesley, MA: Wellesley-
12296 Cambridge Press.
- 12297 Strubell, E., P. Verga, D. Belanger, and A. McCallum (2017). Fast and accurate entity recog-
12298 nition with iterated dilated convolutions. In *Proceedings of Empirical Methods for Natural*
12299 *Language Processing (EMNLP)*.
- 12300 Suchanek, F. M., G. Kasneci, and G. Weikum (2007). Yago: a core of semantic knowledge.
12301 In *Proceedings of the Conference on World-Wide Web (WWW)*, pp. 697–706.

- 12302 Sun, X., T. Matsuzaki, D. Okanohara, and J. Tsujii (2009). Latent variable perceptron algo-
 12303 rithm for structured classification. In *Proceedings of the International Joint Conference on*
 12304 *Artificial Intelligence (IJCAI)*, Volume 9, pp. 1236–1242.
- 12305 Sun, Y., L. Lin, D. Tang, N. Yang, Z. Ji, and X. Wang (2015). Modeling mention, context
 12306 and entity with neural networks for entity disambiguation. In *IJCAI*, pp. 1333–1339.
- 12307 Sundermeyer, M., R. Schlüter, and H. Ney (2012). Lstm neural networks for language
 12308 modeling. In *INTERSPEECH*.
- 12309 Surdeanu, M., J. Tibshirani, R. Nallapati, and C. D. Manning (2012). Multi-instance multi-
 12310 label learning for relation extraction. In *Proceedings of Empirical Methods for Natural Lan-*
 12311 *guage Processing (EMNLP)*, pp. 455–465.
- 12312 Sutskever, I., O. Vinyals, and Q. V. Le (2014). Sequence to sequence learning with neural
 12313 networks. In *Neural Information Processing Systems (NIPS)*, pp. 3104–3112.
- 12314 Sutton, R. S. and A. G. Barto (1998). *Reinforcement learning: An introduction*, Volume 1. MIT
 12315 press Cambridge.
- 12316 Sutton, R. S., D. A. McAllester, S. P. Singh, and Y. Mansour (2000). Policy gradient methods
 12317 for reinforcement learning with function approximation. In *Neural Information Process-*
 12318 *ing Systems (NIPS)*, pp. 1057–1063.
- 12319 Taboada, M., J. Brooke, M. Tofiloski, K. Voll, and M. Stede (2011). Lexicon-based methods
 12320 for sentiment analysis. *Computational linguistics* 37(2), 267–307.
- 12321 Taboada, M. and W. C. Mann (2006). Rhetorical structure theory: Looking back and mov-
 12322 ing ahead. *Discourse studies* 8(3), 423–459.
- 12323 Täckström, O., K. Ganchev, and D. Das (2015). Efficient inference and structured learning
 12324 for semantic role labeling. *Transactions of the Association for Computational Linguistics* 3,
 12325 29–41.
- 12326 Täckström, O., R. McDonald, and J. Uszkoreit (2012). Cross-lingual word clusters for
 12327 direct transfer of linguistic structure. In *Proceedings of the North American Chapter of the*
 12328 *Association for Computational Linguistics (NAACL)*, pp. 477–487.
- 12329 Tang, D., B. Qin, and T. Liu (2015). Document modeling with gated recurrent neural net-
 12330 work for sentiment classification. In *Proceedings of Empirical Methods for Natural Language*
 12331 *Processing (EMNLP)*, pp. 1422–1432.
- 12332 Taskar, B., C. Guestrin, and D. Koller (2003). Max-margin markov networks. In *Neural*
 12333 *Information Processing Systems (NIPS)*.

- 12334 Tausczik, Y. R. and J. W. Pennebaker (2010). The psychological meaning of words: LIWC
12335 and computerized text analysis methods. *Journal of Language and Social Psychology* 29(1),
12336 24–54.
- 12337 Teh, Y. W. (2006). A hierarchical bayesian language model based on pitman-yor processes.
12338 In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 985–992.
- 12339 Tesnière, L. (1966). *Éléments de syntaxe structurale* (second ed.). Paris: Klincksieck.
- 12340 Teufel, S., J. Carletta, and M. Moens (1999). An annotation scheme for discourse-level
12341 argumentation in research articles. In *Proceedings of the European Chapter of the Association
12342 for Computational Linguistics (EACL)*, pp. 110–117.
- 12343 Teufel, S. and M. Moens (2002). Summarizing scientific articles: experiments with relevance
12344 and rhetorical status. *Computational linguistics* 28(4), 409–445.
- 12345 Thomas, M., B. Pang, and L. Lee (2006). Get out the vote: Determining support or opposition
12346 from Congressional floor-debate transcripts. In *Proceedings of Empirical Methods for
12347 Natural Language Processing (EMNLP)*, pp. 327–335.
- 12348 Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal
12349 Statistical Society. Series B (Methodological)*, 267–288.
- 12350 Titov, I. and J. Henderson (2007). Constituent parsing with incremental sigmoid belief
12351 networks. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 632–
12352 639.
- 12353 Toutanova, K., D. Klein, C. D. Manning, and Y. Singer (2003). Feature-rich part-of-speech
12354 tagging with a cyclic dependency network. In *Proceedings of the North American Chapter
12355 of the Association for Computational Linguistics (NAACL)*.
- 12356 Trivedi, R. and J. Eisenstein (2013). Discourse connectors for latent subjectivity in senti-
12357 ment analysis. In *Proceedings of the North American Chapter of the Association for Compu-
12358 tational Linguistics (NAACL)*, pp. 808–813.
- 12359 Tromble, R. W. and J. Eisner (2006). A fast finite-state relaxation method for enforcing
12360 global constraints on sequence decoding. In *Proceedings of the North American Chapter of
12361 the Association for Computational Linguistics (NAACL)*, pp. 423.
- 12362 Tschantaridis, I., T. Hofmann, T. Joachims, and Y. Altun (2004). Support vector machine
12363 learning for interdependent and structured output spaces. In *Proceedings of the twenty-
12364 first international conference on Machine learning*, pp. 104. ACM.
- 12365 Tsvetkov, Y., M. Faruqui, W. Ling, G. Lample, and C. Dyer (2015). Evaluation of word
12366 vector representations by subspace alignment. In *Proceedings of Empirical Methods for
12367 Natural Language Processing (EMNLP)*, pp. 2049–2054.

- 12368 Tu, Z., Z. Lu, Y. Liu, X. Liu, and H. Li (2016). Modeling coverage for neural machine
12369 translation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 76–
12370 85.
- 12371 Turian, J., L. Ratinov, and Y. Bengio (2010). Word representations: a simple and general
12372 method for semi-supervised learning. In *Proceedings of the Association for Computational
12373 Linguistics (ACL)*, pp. 384–394.
- 12374 Turing, A. M. (2009). Computing machinery and intelligence. In *Parsing the Turing Test*,
12375 pp. 23–65. Springer.
- 12376 Turney, P. D. and P. Pantel (2010). From frequency to meaning: Vector space models of
12377 semantics. *Journal of Artificial Intelligence Research* 37, 141–188.
- 12378 Tutin, A. and R. Kittredge (1992). Lexical choice in context: generating procedural texts.
12379 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.
12380 763–769.
- 12381 Twain, M. (1997). *A Tramp Abroad*. New York: Penguin.
- 12382 Tzeng, E., J. Hoffman, T. Darrell, and K. Saenko (2015). Simultaneous deep transfer across
12383 domains and tasks. In *Proceedings of the IEEE International Conference on Computer Vision*,
12384 pp. 4068–4076.
- 12385 Usunier, N., D. Buffoni, and P. Gallinari (2009). Ranking with ordered weighted pairwise
12386 classification. In *Proceedings of the International Conference on Machine Learning (ICML)*,
12387 pp. 1057–1064.
- 12388 Uthus, D. C. and D. W. Aha (2013). The ubuntu chat corpus for multiparicipant chat
12389 analysis. In *AAAI Spring Symposium: Analyzing Microtext*, Volume 13, pp. 01.
- 12390 Utiyama, M. and H. Isahara (2001). A statistical model for domain-independent text seg-
12391 mentation. In *Proceedings of the 39th Annual Meeting on Association for Computational
12392 Linguistics*, pp. 499–506. Association for Computational Linguistics.
- 12393 Utiyama, M. and H. Isahara (2007). A comparison of pivot methods for phrase-based
12394 statistical machine translation. In *Human Language Technologies 2007: The Conference of
12395 the North American Chapter of the Association for Computational Linguistics; Proceedings of
12396 the Main Conference*, pp. 484–491.
- 12397 Uzuner, Ö., X. Zhang, and T. Sibanda (2009). Machine learning and rule-based approaches
12398 to assertion classification. *Journal of the American Medical Informatics Association* 16(1),
12399 109–115.

- 12400 Vadas, D. and J. R. Curran (2011). Parsing noun phrases in the penn treebank. *Computational Linguistics* 37(4), 753–809.
- 12402 Van Eynde, F. (2006). NP-internal agreement and the structure of the noun phrase. *Journal of Linguistics* 42(1), 139–186.
- 12404 Van Gael, J., A. Vlachos, and Z. Ghahramani (2009). The infinite hmm for unsupervised pos tagging. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 678–687.
- 12407 Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017). Attention is all you need. In *Neural Information Processing Systems (NIPS)*, pp. 6000–6010.
- 12410 Velldal, E., L. Øvrelid, J. Read, and S. Oepen (2012). Speculation and negation: Rules, rankers, and the role of syntax. *Computational linguistics* 38(2), 369–410.
- 12412 Versley, Y. (2011). Towards finer-grained tagging of discourse connectives. In *Proceedings of the Workshop Beyond Semantics: Corpus-based Investigations of Pragmatic and Discourse Phenomena*, pp. 2–63.
- 12415 Vilain, M., J. Burger, J. Aberdeen, D. Connolly, and L. Hirschman (1995). A model-theoretic coreference scoring scheme. In *Proceedings of the 6th conference on Message understanding*, pp. 45–52. Association for Computational Linguistics.
- 12418 Vincent, P., H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research* 11(Dec), 3371–3408.
- 12421 Vincze, V., G. Szarvas, R. Farkas, G. Móra, and J. Csirik (2008). The bioscope corpus: biomedical texts annotated for uncertainty, negation and their scopes. *BMC bioinformatics* 9(11), S9.
- 12424 Vinyals, O., A. Toshev, S. Bengio, and D. Erhan (2015). Show and tell: A neural image caption generator. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pp. 3156–3164. IEEE.
- 12427 Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory* 13(2), 260–269.
- 12429 Voll, K. and M. Taboada (2007). Not all words are created equal: Extracting semantic orientation as a function of adjective relevance. In *Proceedings of Australian Conference on Artificial Intelligence*.

- 12432 Wager, S., S. Wang, and P. S. Liang (2013). Dropout training as adaptive regularization. In
 12433 *Neural Information Processing Systems (NIPS)*, pp. 351–359.
- 12434 Wainwright, M. J. and M. I. Jordan (2008). Graphical models, exponential families, and
 12435 variational inference. *Foundations and Trends® in Machine Learning* 1(1-2), 1–305.
- 12436 Walker, M. A. (2000). An application of reinforcement learning to dialogue strategy selec-
 12437 tion in a spoken dialogue system for email. *Journal of Artificial Intelligence Research* 12,
 12438 387–416.
- 12439 Walker, M. A., J. E. Cahn, and S. J. Whittaker (1997). Improvising linguistic style: Social
 12440 and affective bases for agent personality. In *Proceedings of the first international conference*
 12441 on *Autonomous agents*, pp. 96–105. ACM.
- 12442 Wang, C., N. Xue, and S. Pradhan (2015). A Transition-based Algorithm for AMR Parsing.
 12443 In *Proceedings of the North American Chapter of the Association for Computational Linguistics*
 12444 (*NAACL*), pp. 366–375.
- 12445 Wang, H., T. Onishi, K. Gimpel, and D. McAllester (2017). Emergent predication structure
 12446 in hidden state vectors of neural readers. In *Proceedings of the 2nd Workshop on Represen-*
 12447 *tation Learning for NLP*, pp. 26–36.
- 12448 Weaver, W. (1955). Translation. *Machine translation of languages* 14, 15–23.
- 12449 Webber, B. (2004, sep). D-LTAG: extending lexicalized TAG to discourse. *Cognitive Sci-
 12450 ence* 28(5), 751–779.
- 12451 Webber, B., M. Egg, and V. Kordoni (2012). Discourse structure and language technology.
 12452 *Journal of Natural Language Engineering* 1.
- 12453 Webber, B. and A. Joshi (2012). Discourse structure and computation: past, present and
 12454 future. In *Proceedings of the ACL-2012 Special Workshop on Rediscovering 50 Years of Dis-
 12455 coveries*, pp. 42–54. Association for Computational Linguistics.
- 12456 Wei, G. C. and M. A. Tanner (1990). A monte carlo implementation of the em algorithm
 12457 and the poor man’s data augmentation algorithms. *Journal of the American Statistical
 12458 Association* 85(411), 699–704.
- 12459 Weinberger, K., A. Dasgupta, J. Langford, A. Smola, and J. Attenberg (2009). Feature
 12460 hashing for large scale multitask learning. In *Proceedings of the International Conference
 12461 on Machine Learning (ICML)*, pp. 1113–1120.
- 12462 Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language
 12463 communication between man and machine. *Communications of the ACM* 9(1), 36–45.

- 12464 Wellner, B. and J. Pustejovsky (2007). Automatically identifying the arguments of dis-
12465 course connectives. In *Proceedings of Empirical Methods for Natural Language Processing*
12466 (*EMNLP*), pp. 92–101.
- 12467 Wen, T.-H., M. Gasic, N. Mrkšić, P.-H. Su, D. Vandyke, and S. Young (2015). Semantically
12468 conditioned lstm-based natural language generation for spoken dialogue systems. In
12469 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1711–1721.
- 12470 Weston, J., S. Bengio, and N. Usunier (2011). Wsabie: Scaling up to large vocabulary image
12471 annotation. In *IJCAI*, Volume 11, pp. 2764–2770.
- 12472 Wiebe, J., T. Wilson, and C. Cardie (2005). Annotating expressions of opinions and emo-
12473 tions in language. *Language resources and evaluation* 39(2), 165–210.
- 12474 Wieting, J., M. Bansal, K. Gimpel, and K. Livescu (2015). Towards universal paraphrastic
12475 sentence embeddings. *arXiv preprint arXiv:1511.08198*.
- 12476 Wieting, J., M. Bansal, K. Gimpel, and K. Livescu (2016). CHARAGRAM: Embedding
12477 words and sentences via character n-grams. In *Proceedings of Empirical Methods for Nat-*
12478 *ural Language Processing (EMNLP)*, pp. 1504–1515.
- 12479 Williams, J. D. and S. Young (2007). Partially observable markov decision processes for
12480 spoken dialog systems. *Computer Speech & Language* 21(2), 393–422.
- 12481 Williams, P., R. Sennrich, M. Post, and P. Koehn (2016). Syntax-based statistical machine
12482 translation. *Synthesis Lectures on Human Language Technologies* 9(4), 1–208.
- 12483 Wilson, T., J. Wiebe, and P. Hoffmann (2005). Recognizing contextual polarity in phrase-
12484 level sentiment analysis. In *Proceedings of Empirical Methods for Natural Language Pro-*
12485 *cessing (EMNLP)*, pp. 347–354.
- 12486 Winograd, T. (1972). Understanding natural language. *Cognitive psychology* 3(1), 1–191.
- 12487 Wiseman, S., A. M. Rush, and S. M. Shieber (2016). Learning global features for corefer-
12488 ence resolution. In *Proceedings of the North American Chapter of the Association for Compu-*
12489 *tational Linguistics (NAACL)*, pp. 994–1004.
- 12490 Wiseman, S., S. Shieber, and A. Rush (2017). Challenges in data-to-document generation.
12491 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 2253–
12492 2263.
- 12493 Wiseman, S. J., A. M. Rush, S. M. Shieber, and J. Weston (2015). Learning anaphoricity and
12494 antecedent ranking features for coreference resolution. In *Proceedings of the Association*
12495 *for Computational Linguistics (ACL)*.

- 12496 Wolf, F. and E. Gibson (2005). Representing discourse coherence: A corpus-based study.
 12497 *Computational Linguistics* 31(2), 249–287.
- 12498 Wolfe, T., M. Dredze, and B. Van Durme (2017). Pocket knowledge base population. In
 12499 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 305–310.
- 12500 Wong, Y. W. and R. Mooney (2007). Generation by inverting a semantic parser that uses
 12501 statistical machine translation. In *Proceedings of the North American Chapter of the Associa-*
 12502 *tion for Computational Linguistics (NAACL)*, pp. 172–179.
- 12503 Wong, Y. W. and R. J. Mooney (2006). Learning for semantic parsing with statistical ma-
 12504 chine translation. In *Proceedings of the North American Chapter of the Association for Com-*
 12505 *putational Linguistics (NAACL)*, pp. 439–446.
- 12506 Wu, B. Y. and K.-M. Chao (2004). *Spanning trees and optimization problems*. CRC Press.
- 12507 Wu, D. (1997). Stochastic inversion transduction grammars and bilingual parsing of par-
 12508 allel corpora. *Computational linguistics* 23(3), 377–403.
- 12509 Wu, F. and D. S. Weld (2010). Open information extraction using wikipedia. In *Proceedings*
 12510 *of the Association for Computational Linguistics (ACL)*, pp. 118–127.
- 12511 Wu, X., R. Ward, and L. Bottou (2018). Wngrad: Learn the learning rate in gradient de-
 12512 scent. *arXiv preprint arXiv:1803.02865*.
- 12513 Wu, Y., M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao,
 12514 Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws,
 12515 Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young,
 12516 J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean (2016).
 12517 Google’s neural machine translation system: Bridging the gap between human and ma-
 12518 chine translation. *CoRR abs/1609.08144*.
- 12519 Xia, F. (2000). The part-of-speech tagging guidelines for the penn chinese treebank (3.0).
 12520 Technical report, University of Pennsylvania Institute for Research in Cognitive Science.
- 12521 Xu, K., J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio
 12522 (2015). Show, attend and tell: Neural image caption generation with visual attention.
 12523 In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 2048–2057.
- 12524 Xu, W., X. Liu, and Y. Gong (2003). Document clustering based on non-negative matrix
 12525 factorization. In *SIGIR*, pp. 267–273. ACM.
- 12526 Xu, Y., L. Mou, G. Li, Y. Chen, H. Peng, and Z. Jin (2015). Classifying relations via long
 12527 short term memory networks along shortest dependency paths. In *Proceedings of Empir-*
 12528 *ical Methods for Natural Language Processing (EMNLP)*, pp. 1785–1794.

- 12529 Xuan Bach, N., N. L. Minh, and A. Shimazu (2012). A reranking model for discourse seg-
12530 mentation using subtree features. In *Proceedings of the Special Interest Group on Discourse*
12531 *and Dialogue (SIGDIAL)*.
- 12532 Xue, N. et al. (2003). Chinese word segmentation as character tagging. *Computational*
12533 *Linguistics and Chinese Language Processing* 8(1), 29–48.
- 12534 Xue, N., H. T. Ng, S. Pradhan, R. Prasad, C. Bryant, and A. T. Rutherford (2015). The
12535 CoNLL-2015 shared task on shallow discourse parsing. In *Proceedings of the Conference*
12536 *on Natural Language Learning (CoNLL)*.
- 12537 Xue, N., H. T. Ng, S. Pradhan, A. Rutherford, B. L. Webber, C. Wang, and H. Wang (2016).
12538 Conll 2016 shared task on multilingual shallow discourse parsing. In *CoNLL Shared*
12539 *Task*, pp. 1–19.
- 12540 Yamada, H. and Y. Matsumoto (2003). Statistical dependency analysis with support vector
12541 machines. In *Proceedings of IWPT*, Volume 3, pp. 195–206.
- 12542 Yamada, K. and K. Knight (2001). A syntax-based statistical translation model. In *Proceed-*
12543 *ings of the 39th Annual Meeting on Association for Computational Linguistics*, pp. 523–530.
12544 Association for Computational Linguistics.
- 12545 Yang, B. and C. Cardie (2014). Context-aware learning for sentence-level sentiment anal-
12546 ysis with posterior regularization. In *Proceedings of the Association for Computational Lin-*
12547 *guistics (ACL)*.
- 12548 Yang, Y., M.-W. Chang, and J. Eisenstein (2016). Toward socially-infused information ex-
12549 traction: Embedding authors, mentions, and entities. In *Proceedings of Empirical Methods*
12550 *for Natural Language Processing (EMNLP)*.
- 12551 Yang, Y. and J. Eisenstein (2013). A log-linear model for unsupervised text normalization.
12552 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12553 Yang, Y. and J. Eisenstein (2015). Unsupervised multi-domain adaptation with feature em-
12554 beddings. In *Proceedings of the North American Chapter of the Association for Computational*
12555 *Linguistics (NAACL)*.
- 12556 Yang, Y., W.-t. Yih, and C. Meek (2015). WikiQA: A challenge dataset for open-domain
12557 question answering. In *Proceedings of Empirical Methods for Natural Language Processing*
12558 *(EMNLP)*, pp. 2013–2018.
- 12559 Yannakoudakis, H., T. Briscoe, and B. Medlock (2011). A new dataset and method for
12560 automatically grading esol texts. In *Proceedings of the 49th Annual Meeting of the Associa-*
12561 *tion for Computational Linguistics: Human Language Technologies-Volume 1*, pp. 180–189.
12562 Association for Computational Linguistics.

- 12563 Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised meth-
 12564 ods. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 189–196.
 12565 Association for Computational Linguistics.
- 12566 Yee, L. C. and T. Y. Jones (2012, March). Apple ceo in china mission to clear up problems.
 12567 *Reuters*. retrieved on March 26, 2017.
- 12568 Yi, Y., C.-Y. Lai, S. Petrov, and K. Keutzer (2011, October). Efficient parallel cky parsing on
 12569 gpus. In *Proceedings of the 12th International Conference on Parsing Technologies*, Dublin,
 12570 Ireland, pp. 175–185. Association for Computational Linguistics.
- 12571 Yu, C.-N. J. and T. Joachims (2009). Learning structural svms with latent variables. In
 12572 *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1169–1176.
- 12573 Yu, F. and V. Koltun (2016). Multi-scale context aggregation by dilated convolutions. In
 12574 *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 12575 Zaidan, O. F. and C. Callison-Burch (2011). Crowdsourcing translation: Professional qual-
 12576 ity from non-professionals. In *Proceedings of the Association for Computational Linguistics*
 12577 (*ACL*), pp. 1220–1229.
- 12578 Zaremba, W., I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv*
 12579 preprint *arXiv:1409.2329*.
- 12580 Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint*
 12581 *arXiv:1212.5701*.
- 12582 Zelenko, D., C. Aone, and A. Richardella (2003). Kernel methods for relation extraction.
 12583 *The Journal of Machine Learning Research* 3, 1083–1106.
- 12584 Zelle, J. M. and R. J. Mooney (1996). Learning to parse database queries using induc-
 12585 tive logic programming. In *Proceedings of the National Conference on Artificial Intelligence*
 12586 (*AAAI*), pp. 1050–1055.
- 12587 Zeng, D., K. Liu, S. Lai, G. Zhou, and J. Zhao (2014). Relation classification via convolu-
 12588 tional deep neural network. In *Proceedings of the International Conference on Computational*
 12589 *Linguistics (COLING)*, pp. 2335–2344.
- 12590 Zettlemoyer, L. S. and M. Collins (2005). Learning to map sentences to logical form: Struc-
 12591 tured classification with probabilistic categorial grammars. In *Proceedings of UAI*.
- 12592 Zhang, X., J. Zhao, and Y. LeCun (2015). Character-level convolutional networks for text
 12593 classification. In *Neural Information Processing Systems (NIPS)*, pp. 649–657.

- 12594 Zhang, Y. and S. Clark (2008). A tale of two parsers: investigating and combining graph-
12595 based and transition-based dependency parsing using beam-search. In *Proceedings of*
12596 *Empirical Methods for Natural Language Processing (EMNLP)*, pp. 562–571.
- 12597 Zhang, Y., T. Lei, R. Barzilay, T. Jaakkola, and A. Globerson (2014). Steps to excellence:
12598 Simple inference with refined scoring of dependency trees. In *Proceedings of the Associa-*
12599 *tion for Computational Linguistics (ACL)*, pp. 197–207.
- 12600 Zhang, Y. and J. Nivre (2011). Transition-based dependency parsing with rich non-local
12601 features. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 188–193.
- 12602 Zhou, J. and W. Xu (2015). End-to-end learning of semantic role labeling using recurrent
12603 neural networks. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
12604 1127–1137.
- 12605 Zhu, J., Z. Nie, X. Liu, B. Zhang, and J.-R. Wen (2009). Statsnowball: a statistical approach
12606 to extracting entity relationships. In *Proceedings of the Conference on World-Wide Web*
12607 (*WWW*), pp. 101–110.
- 12608 Zhu, X., Z. Ghahramani, and J. D. Lafferty (2003). Semi-supervised learning using gaus-
12609 sian fields and harmonic functions. In *Proceedings of the International Conference on Ma-*
12610 *chine Learning (ICML)*, pp. 912–919.
- 12611 Zhu, X. and A. B. Goldberg (2009). Introduction to semi-supervised learning. *Synthesis*
12612 *lectures on artificial intelligence and machine learning* 3(1), 1–130.
- 12613 Zipf, G. K. (1949). Human behavior and the principle of least effort.
- 12614 Zirn, C., M. Niepert, H. Stuckenschmidt, and M. Strube (2011). Fine-grained sentiment
12615 analysis with structural features. In *IJCNLP*, Chiang Mai, Thailand, pp. 336–344.
- 12616 Zou, W. Y., R. Socher, D. Cer, and C. D. Manning (2013). Bilingual word embeddings
12617 for phrase-based machine translation. In *Proceedings of Empirical Methods for Natural*
12618 *Language Processing (EMNLP)*, pp. 1393–1398.

12619 **Index**

- 12620 *K*-means, 106
12621 α -conversion, 296
12622 β -conversion, 293
12623 β -reduction, 293
12624 *n*-gram language models, 204
12625 *n*-gram, 38
12626 *F*-MEASURE, 92
12627 BLEU, 435
12628 WordNet, 84

12629 ablation test, 94
12630 absolute discounting, 140
12631 Abstract Meaning Representation
 (AMR), 309, 323
12633 abstractive summarization, 397, 464
12634 accepting path, 199
12635 accuracy, 36, 91
12636 action (reinforcement learning), 373
12637 active learning, 128
12638 AdaDelta (online optimization), 73
12639 AdaGrad, 53
12640 AdaGrad (online optimization), 73
12641 Adam (online optimization), 73
12642 adequacy (translation), 435
12643 adjectives, 185
12644 adjuncts (semantics), 308, 312
12645 adpositions, 186
12646 adverbs, 185
12647 adversarial networks, 123
12648 affix (morphology), 201
12649 agenda-based dialogue systems, 468

12650 agenda-based parsing, 280
12651 agent (thematic role), 310
12652 alignment, 325, 434, 439
12653 alignment (in text generation), 459
12654 alignment (text generation), 459
12655 Amazon Mechanical Turk, 100
12656 ambiguity, 216, 224
12657 anaphoric, 368
12658 anchored productions, 238, 244
12659 annealing, 454
12660 antecedent (coreference), 359, 367
12661 antonymy, 84
12662 apophony, 200
12663 arc-eager dependency parsing, 271, 273
12664 arc-factored dependency parsing, 265
12665 arc-standard, 271
12666 arc-standard dependency parsing, 271
12667 area under the curve (AUC), 93
12668 argumentation, 396
12669 argumentation mining, 397
12670 arguments, 405
12671 article (syntax), 190
12672 aspect, 185
12673 attachment ambiguity, 235, 259
12674 attention, 460
12675 attention mechanism, 379, 428, 447
12676 autoencoders, 353
12677 automated theorem provers, 288
12678 automatic differentiation, 68
12679 auxiliary verbs, 186
12680 average mutual information, 341

- 12681 averaged perceptron, 40
 12682 backchannel, 195
 12683 backoff, 140
 12684 backpropagation, 67, 146
 12685 backpropagation through time, 146
 12686 backward recurrence, 173, 174
 12687 backward-looking center, 386
 12688 bag of words, 27
 12689 balanced *F*-MEASURE, 93
 12690 balanced test set, 91
 12691 batch normalization, 72
 12692 batch optimization, 51
 12693 Baum-Welch algorithm, 178
 12694 Bayes' rule, 476
 12695 Bayesian nonparametrics, 113, 254
 12696 beam sampling, 180
 12697 beam search, 273, 372, 451, 452
 12698 Bell number, 370
 12699 best-path algorithm, 204
 12700 bias, 137
 12701 bias (learning theory), 36
 12702 bias-variance tradeoff, 36, 139
 12703 biconvexity, 112
 12704 bidirectional LSTM, 447
 12705 bidirectional recurrent neural network,
 176
 12707 bigrams, 38, 80
 12708 bilexical, 252
 12709 bilexical features, 268
 12710 bilinear product, 338
 12711 binarization (context-free grammar), 217
 12712 binarize, 234
 12713 binomial distribution, 94
 12714 binomial random variable, 480
 12715 binomial test, 94
 12716 BIO notation, 192, 320
 12717 biomedical natural language processing,
 191
 12719 bipartite graph, 327
 12720 bitext, 434
 12721 Bonferroni correction, 97
 12722 boolean semiring, 205
 12723 boosting, 60
 12724 bootstrap samples, 96
 12725 bound variable, 290
 12726 brevity penalty (machine translation),
 436
 12728 Brown clusters, 335
 12729 byte pair encoding, 451
 12730 byte-pair encodings, 351
 12731 c-command, 361
 12732 case marking, 190, 226
 12733 Catalan number, 231
 12734 cataphora, 360
 12735 center embedding, 213
 12736 centering theory, 363, 386
 12737 chain FSA, 211
 12738 chain rule of probability, 476
 12739 chance agreement, 100
 12740 character-level language models, 151
 12741 chart parsing, 232
 12742 chatbots, 470
 12743 Chomsky Normal Form (CNF), 217
 12744 Chu-Liu-Edmonds algorithm, 266
 12745 CKY algorithm, 231, 232
 12746 class imbalance, 91
 12747 classification weights, 27
 12748 cleft, 364
 12749 closed-vocabulary, 151
 12750 closure (regular languages), 198
 12751 cloze question answering, 427
 12752 cluster ranking, 371
 12753 clustering, 106
 12754 co-training, 118
 12755 coarse-to-fine attention, 462
 12756 code switching, 188, 194
 12757 Cohen's Kappa, 100
 12758 coherence, 400
 12759 cohesion, 383
 12760 collapsed Gibbs sampling, 126

- 12761 collective entity linking, 410
 12762 collocation extraction, 352
 12763 collocation features, 85
 12764 combinatorial optimization, 19
 12765 combinatory categorial grammar, 226
 12766 complement clause, 219
 12767 complement event (probability), 474
 12768 complement structure, 235
 12769 composition (CCG), 227
 12770 compositional vector grammars, 395
 12771 compositionality, 18, 21, 349
 12772 computation graph, 60, 67
 12773 computational linguistics (versus natural language processing), 13
 12774 computational social science, 17
 12775 concept (AMR), 323
 12777 conditional independence, 478
 12778 conditional log-likelihood, 64
 12779 conditional probability, 48, 475
 12780 conditional probability distribution, 479
 12781 conditional random field, 170
 12782 conditionally independent, 162
 12783 confidence interval, 96
 12784 configuration (transition-based parsing), 271
 12785 connected (graph theory), 326
 12787 consistency (logic), 291
 12788 constants (logic), 286
 12789 constituents, 218
 12790 constrained optimization, 45, 317
 12791 constraint-driven learning, 128
 12792 constraints, 317
 12793 content selection (text generation), 457
 12794 content words, 186
 12795 context vector (attentional neural translation), 448
 12796 context-free grammars (CFGs), 214
 12798 context-free languages, 213, 214
 12799 context-sensitive languages, 225
 12800 continuous bag-of-words (CBOW), 343
 12801 contradiction, 354
 12802 conversational turns, 195
 12803 convex, 42, 480
 12804 convex optimization, 51
 12805 convexity, 70, 301, 483
 12806 convolutional neural network, 151
 12807 convolutional neural networks, 65, 74, 80, 178, 193, 320, 417
 12808 cooperative principle, 359
 12810 coordinate ascent, 112
 12811 coordinating conjunctions, 186
 12812 coordinating discourse relations, 393
 12813 coordination scope, 235
 12814 copula, 223, 262
 12815 copula verb, 185
 12816 coreference resolution, 355, 359
 12817 coreferent, 359
 12818 cosine similarity, 348, 384
 12819 cost-augmented decoding, 47, 169
 12820 cost-augmented inference, 47
 12821 count, 90
 12822 coverage (summarization), 397
 12823 coverage loss, 465
 12824 critical point, 70, 483
 12825 cross-document coreference resolution, 408
 12826 cross-entropy, 64, 418
 12828 cross-serial dependencies, 225
 12829 cross-validation, 37
 12830 crowdsourcing, 100
 12831 cumulative probability distribution, 95
 12832 dead neurons, 63
 12833 decidability (logic), 291
 12834 decision trees, 60
 12835 deep learning, 59
 12836 deep LSTM, 446
 12837 definiteness, 191
 12838 delta function, 35
 12839 denotation (semantics), 286
 12840 dependency grammar, 259
 12841 dependency graph, 260

- | | | | |
|-------|--|-------|--|
| 12842 | dependency parse, 259 | 12883 | dual decomposition, 320 |
| 12843 | dependency path, 85, 316, 415 | 12884 | dual form, 485 |
| 12844 | dependent, 260 | 12885 | dynamic computation graphs, 68 |
| 12845 | derivation, 271 | 12886 | dynamic oracle, 278 |
| 12846 | derivation (context-free languages), 215 | 12887 | dynamic programming, 157 |
| 12847 | derivation (semantic parsing), 294, 298 | 12888 | dynamic semantics, 303, 388 |
| 12848 | derivational ambiguity, 228 | 12889 | E-step (expectation-maximization), 109 |
| 12849 | derivational morphology, 200 | 12890 | early stopping, 41, 73 |
| 12850 | derivations, 276 | 12891 | early update, 278 |
| 12851 | determiner, 187 | 12892 | easy-first parsing, 281 |
| 12852 | determiner phrase, 221 | 12893 | edit distance, 207 |
| 12853 | deterministic FSA, 200 | 12894 | effective count, 139 |
| 12854 | development set, 37, 91 | 12895 | elementary discourse units, 392 |
| 12855 | dialogue acts, 100, 195, 472 | 12896 | elementwise nonlinearity, 61 |
| 12856 | dialogue management, 468 | 12897 | Elman unit, 145 |
| 12857 | dialogue systems, 135, 467 | 12898 | embedding, 175, 343 |
| 12858 | digital humanities, 17, 79 | 12899 | emission features, 156 |
| 12859 | dilated convolution, 75 | 12900 | emotion, 82 |
| 12860 | dilated convolutions, 193 | 12901 | empirical Bayes, 126 |
| 12861 | Dirichlet distribution, 125 | 12902 | empty string, 198 |
| 12862 | discount, 140 | 12903 | encoder-decoder, 444 |
| 12863 | discourse, 383 | 12904 | encoder-decoder model, 353, 460 |
| 12864 | discourse connectives, 389 | 12905 | ensemble, 323 |
| 12865 | discourse depth, 398 | 12906 | ensemble learning, 446 |
| 12866 | discourse depth tree, 398, 399 | 12907 | ensemble methods, 60 |
| 12867 | discourse parsing, 388 | 12908 | entailment, 291, 354 |
| 12868 | discourse relations, 355 | 12909 | entities, 405 |
| 12869 | discourse segment, 383 | 12910 | entity embeddings, 410 |
| 12870 | discourse sense classification, 390 | 12911 | entity grid, 387 |
| 12871 | discourse unit, 392 | 12912 | entity linking, 359, 405, 407, 418 |
| 12872 | discrete random variable, 478 | 12913 | entropy, 55, 109 |
| 12873 | discriminative learning, 38 | 12914 | estimation, 480 |
| 12874 | disjoint events, 474 | 12915 | EuroParl corpus, 437 |
| 12875 | distant supervision, 128, 420, 421 | 12916 | event, 422 |
| 12876 | distributional, 253, 334 | 12917 | event (probability), 473 |
| 12877 | distributional hypothesis, 333, 334 | 12918 | event coreference, 423 |
| 12878 | distributional semantics, 21 | 12919 | event detection, 423 |
| 12879 | distributional statistics, 85 | 12920 | event semantics, 307 |
| 12880 | document frequency, 409 | 12921 | events, 405 |
| 12881 | domain adaptation, 105, 121 | 12922 | evidentiality, 190, 425 |
| 12882 | dropout, 69, 147 | | |

- 12923 exchange clustering, 342
 12924 existential quantifier, 290
 12925 expectation, 479
 12926 expectation maximization, 107, 141
 12927 expectation semiring, 213
 12928 expectation-maximization, in machine translation, 441
 12930 explicit semantic analysis, 336
 12931 exploding gradients, 147
 12932 extra-propositional semantics, 424
 12933 extractive question-answering, 428
 12934 extractive summarization, 397
 12935 extraposition, 364
 12936 extrinsic evaluation, 149
- 12937 factoid questions, 327
 12938 factoids, 426
 12939 factor graph, 171
 12940 factuality, 425
 12941 false discovery rate, 97
 12942 False negative, 91
 12943 False positive, 91
 12944 false positive, 477
 12945 false positive rate, 93, 476
 12946 feasible set, 317
 12947 feature co-adaptation, 69
 12948 feature function, 28, 37
 12949 feature hashing, 90
 12950 feature noising, 70
 12951 feature selection, 54
 12952 features, 18
 12953 feedforward neural network, 62
 12954 fine-tuned word embeddings, 349
 12955 finite state acceptor (FSA), 199
 12956 finite state automata, 199
 12957 finite state composition, 210
 12958 finite state transducers, 202, 207
 12959 first-order logic, 289
 12960 fluency (translation), 435
 12961 fluent, 135
 12962 focus, 325
- 12963 formal language theory, 197
 12964 forward recurrence, 172
 12965 forward variable, 174
 12966 forward variables, 172
 12967 forward-backward algorithm, 173, 212, 245
 12968 forward-looking centers, 386
 12969 frame, 467
 12970 frame elements, 313
 12972 FrameNet, 313
 12973 frames, 312
 12974 free variable, 289
 12975 Frobenius norm, 69
 12976 function (first-order logic), 290
 12977 function words, 186
 12978 functional margin, 45
 12979 functional segmentation, 383, 385
- 12980 garden path sentence, 154
 12981 gate (neural networks), 64, 447
 12982 gazetteer, 415
 12983 gazetteers, 366
 12984 gazetteers, 192
 12985 generalization, 41
 12986 generalized linear models, 55
 12987 generative model, 31, 241
 12988 generative models, 371
 12989 generative process, 141
 12990 generic referents, 364
 12991 geometric margin, 45
 12992 Gibbs sampling, 125, 411
 12993 gloss, 135, 187, 435, 442
 12994 government and binding theory, 361
 12995 gradient, 43
 12996 gradient clipping, 72
 12997 gradient descent, 51
 12998 Gram matrix, 415
 12999 grammar induction, 246
 13000 grammaticality, 400
 13001 graph-based dependency parsing, 264
 13002 graphical model, 162

- | | | | |
|-------|---|-------|--|
| 13003 | graphics processing units (GPUs), 178, | 13042 | inference, 155 |
| 13004 | 193 | 13043 | inference (logic), 285 |
| 13005 | grid search, 36 | 13044 | inference rules, 288 |
| 13006 | Hamming cost, 169 | 13045 | inflection point, 484 |
| 13007 | Hansards corpus, 437 | 13046 | inflectional affixes, 88 |
| 13008 | hanzi, 87 | 13047 | inflectional morphology, 185, 200, 208 |
| 13009 | hard expectation-maximization, 112 | 13048 | information extraction, 405 |
| 13010 | hard tanh, 321 | 13049 | information retrieval, 17, 419 |
| 13011 | head, 260, 415 | 13050 | initiative (dialogue systems), 468 |
| 13012 | head rules, 259 | 13051 | input word embeddings, 145 |
| 13013 | head word, 218, 259, 365 | 13052 | inside recurrence, 242, 245 |
| 13014 | head words, 249 | 13053 | inside-outside algorithm, 245, 253 |
| 13015 | hedging, 425 | 13054 | instance (AMR), 323 |
| 13016 | held-out data, 149 | 13055 | instance labels, 30 |
| 13017 | Hessian matrix, 52 | 13056 | integer linear program, 430, 466 |
| 13018 | hidden Markov models, 162 | 13057 | integer linear programming, 317, 370, |
| 13019 | hidden variable perceptron, 213 | 13058 | 398, 411 |
| 13020 | hierarchical clustering, 339 | 13059 | inter-annotator agreement, 100 |
| 13021 | hierarchical recurrent network, 470 | 13060 | interjections, 185 |
| 13022 | hierarchical softmax, 145, 345 | 13061 | interlingua, 434 |
| 13023 | hierarchical topic segmentation, 385 | 13062 | interpolated n -gram language model, |
| 13024 | highway network, 64 | 13063 | 205 |
| 13025 | hinge loss, 42 | 13064 | interpolation, 141 |
| 13026 | homonym, 83 | 13065 | interval algebra, 423 |
| 13027 | human computation, 101 | 13066 | intrinsic evaluation, 149 |
| 13028 | hypergraph, 396 | 13067 | inverse document frequency, 409 |
| 13029 | hyperparameter, 36 | 13068 | inverse relation (AMR), 325 |
| 13030 | hyponymy, 85 | 13069 | inversion (finite state), 209 |
| 13031 | illocutionary force, 195 | 13070 | irrealis, 80 |
| 13032 | implicit discourse relations, 390 | 13071 | Jeffreys-Perks law, 139 |
| 13033 | importance sampling, 455 | 13072 | Jensen's inequality, 109 |
| 13034 | importance score, 455 | 13073 | joint probabilities, 479 |
| 13035 | incremental expectation maximization, | 13074 | joint probability, 30, 48 |
| 13036 | 112 | 13075 | Kalman smoother, 180 |
| 13037 | incremental perceptron, 278, 372 | 13076 | Katz backoff, 140 |
| 13038 | independent and identically distributed | 13077 | kernel function, 415 |
| 13039 | (IID), 30 | 13078 | kernel methods, 60 |
| 13040 | indicator function, 35 | 13079 | kernel support vector machine, 60, 416 |
| 13041 | Indicator random variables, 478 | 13080 | Kleene star, 198 |

- 13081 knapsack problem, 398
 13082 knowledge base, 405
 13083 knowledge base population, 418
 13084 L-BFGS, 52
 13085 label bias problem, 277
 13086 label propagation, 120, 130
 13087 labeled dependencies, 262
 13088 labeled precision, 236
 13089 labeled recall, 236
 13090 Lagrange multiplier, 485
 13091 Lagrangian, 485
 13092 lambda calculus, 292
 13093 lambda expressions, 293
 13094 language model, 136
 13095 language models, 16
 13096 Laplace smoothing, 36, 139
 13097 large margin classification, 44
 13098 latent conditional random fields, 301
 13099 latent Dirichlet allocation, 385
 13100 latent semantic analysis, 336, 338
 13101 latent variable, 108, 212, 421, 434
 13102 latent variable perceptron, 301, 368
 13103 latent variables, 300
 13104 layer normalization, 73, 449
 13105 leaky ReLU, 63
 13106 learning to search, 259, 279, 374
 13107 least squares, 82
 13108 leave-one-out, 37
 13109 lemma, 83
 13110 lemma (lexical semantics), 208
 13111 lemmatization, 88
 13112 Levenshtein edit distance, 207
 13113 lexical entry, 294
 13114 lexical features, 59
 13115 lexical semantics, 83
 13116 lexical unit (frame semantics), 312
 13117 lexicalization, 249
 13118 lexicalization (text generation), 457
 13119 lexicalized tree-adjoining grammar for discourse (D-LTAG), 389
 13121 lexicon, 294
 13122 lexicon (CCG), 227
 13123 lexicon-based classification, 82
 13124 lexicon-based sentiment analysis, 80
 13125 Lidstone smoothing, 139
 13126 light verb, 325
 13127 likelihood, 476
 13128 linear regression, 82
 13129 linear separability, 39
 13130 linearization, 467
 13131 literal character, 198
 13132 local minimum, 484
 13133 local optimum, 112
 13134 locally-normalized objective, 277
 13135 log-bilinear language model, 350
 13136 logistic function, 55
 13137 logistic loss, 49
 13138 logistic regression, 48, 55
 13139 Long short-term memories, 145
 13140 long short-term memories, 322
 13141 long short-term memory, 147
 13142 long short-term memory (LSTM), 64, 189, 445
 13143 lookup layer, 65, 145
 13145 loss function, 41
 13146 LSTM, 147
 13147 LSTM-CRF, 177, 322
 13148 machine learning, 14
 13149 machine reading, 427
 13150 machine translation, 135
 13151 Macro F-MEASURE, 92
 13152 macro-reading, 406
 13153 margin, 39, 44
 13154 marginal probability distribution, 479
 13155 marginal relevance, 465
 13156 marginalize, 475
 13157 markable, 366
 13158 Markov assumption, 162
 13159 Markov blanket, 162

- 13160 Markov Chain Monte Carlo (MCMC), 113, 125, 180
 13161 Markov decision process, 468
 13163 Markov random fields, 170
 13164 matrix-tree theorem, 270
 13165 max pooling, 321
 13166 max-margin Markov network, 169
 13167 max-product algorithm, 165
 13168 maximum a posteriori, 36, 481
 13169 maximum conditional likelihood, 48
 13170 maximum entropy, 55
 13171 maximum likelihood, 480
 13172 maximum likelihood estimate, 34
 13173 maximum likelihood estimation, 30
 13174 maximum spanning tree, 266
 13175 McNemar's test, 94
 13176 meaning representation, 285
 13177 membership problem, 197
 13178 memory cell (LSTM), 147
 13179 mention (coreference resolution), 359
 13180 mention (entity), 405
 13181 mention (information extraction), 407
 13182 mention ranking, 368
 13183 mention-pair model, 367
 13184 meronymy, 85
 13185 meteor, 437
 13186 method of moments, 126
 13187 micro F -MEASURE, 92
 13188 micro-reading, 406
 13189 mildly context-sensitive languages, 225
 13190 minibatch, 52
 13191 minimization (FSA), 202
 13192 minimum error-rate training (MERT), 453
 13194 minimum risk training, 454
 13195 mixed-initiative, 468
 13196 modality, 424
 13197 model, 19
 13198 model builder, 291
 13199 model checker, 291
 13200 model-theoretic semantics, 286
 13201 modeling (machine learning), 51
 13202 modifier (dependency grammar), 260
 13203 modifier scope, 235
 13204 modus ponens, 288
 13205 moment-matching, 55
 13206 monomorphemic, 202
 13207 morphemes, 17, 151, 201
 13208 morphological analysis, 208
 13209 morphological generation, 208
 13210 morphological segmentation, 167
 13211 morphology, 89, 166, 200, 349, 450
 13212 morphosyntactic, 184
 13213 morphosyntactic attributes, 188
 13214 morphotactic, 201
 13215 multi-document summarization, 466
 13216 multi-task learning, 321
 13217 multi-view learning, 118
 13218 multilayer perceptron, 62
 13219 multinomial distribution, 32
 13220 multinomial naïve Bayes, 32
 13221 multiple instance learning, 128, 420
 13222 multitask learning, 128
 13223 Naïve Bayes, 31
 13224 name dictionary, 408
 13225 named entities, 191
 13226 named entity linking, 407
 13227 named entity recognition, 177, 405, 407
 13228 named entity types, 407
 13229 narrow convolution, 75
 13230 nearest-neighbor, 60, 416
 13231 negation, 80, 424
 13232 negative sampling, 345, 346, 413
 13233 Neo-Davidsonian event semantics, 308
 13234 neural attention, 446
 13235 neural machine translation, 434
 13236 neural networks, 59, 144
 13237 NIL entity, 408
 13238 noise-contrastive estimation, 145
 13239 noisy channel model, 136, 438
 13240 nominal modifier, 221

- 13241 nominals, 359
 13242 nominals (coreference), 366
 13243 non-convex, 42
 13244 non-core roles (AMR), 324
 13245 non-terminals (context-free grammars),
 215
 13246 normalization, 88
 13248 noun phrase, 14, 218
 13249 nouns, 184
 13250 NP-hard, 54, 411
 13251 nuclearity (RST), 393
 13252 nucleus (RST), 393
 13253 null hypothesis, 94
 13254 numeral (part of speech), 187
 13255 numerical optimization, 20
 13256 offset feature, 29
 13257 one-dimensional convolution, 75
 13258 one-hot, 380
 13259 one-hot vector, 64
 13260 one-tailed p-value, 95
 13261 one-versus-all multiclass classification,
 416
 13262 one-versus-one multiclass classification,
 416
 13263 online expectation maximization, 112
 13266 online learning, 39, 52
 13267 ontology, 20
 13268 open information extraction, 421
 13269 open word classes, 184
 13270 opinion polarity, 79
 13271 oracle, 276, 327
 13272 oracle (learning to search), 374
 13273 orthography, 202, 210
 13274 orthonormal matrix, 71
 13275 out-of-vocabulary words, 189
 13276 outside recurrence, 243, 245
 13277 overfit, 36
 13278 overfitting, 40
 13279 overgenerate, 297
 13280 overgeneration, 209, 219
 13281 parallel corpora, 437
 13282 parameters, 480
 13283 paraphrase, 354
 13284 parent annotation, 248
 13285 parsing, 215
 13286 part-of-speech, 183
 13287 part-of-speech tagging, 153
 13288 partially observable Markov decision
 process (POMDP), 470
 13289 partially supervised learning, 253
 13291 particle (part-of-speech), 187, 223
 13292 particle versus preposition, 235
 13293 partition, 475
 13294 partition function, 172
 13295 parts-of-speech, 17
 13296 passive-aggressive, 485
 13297 path (finite state automata), 199
 13298 Penn Discourse Treebank (PDTB), 389
 13299 Penn Treebank, 150, 167, 188, 218, 243
 13300 perceptron, 39
 13301 perplexity, 150
 13302 phonology, 202
 13303 phrase (syntax), 218
 13304 phrase-structure grammar, 218
 13305 pivot features, 122
 13306 planning, 458
 13307 pleonastic, 364
 13308 pointwise mutual information, 338
 13309 policy, 373, 469
 13310 policy (search), 278
 13311 policy gradient, 374
 13312 polysemous, 84
 13313 pooling, 380
 13314 pooling (convolution), 75, 380, 460
 13315 pooling (in convolutional neural
 networks), 322
 13317 positional encodings, 449
 13318 positive pointwise mutual information,
 339
 13319 posterior, 476
 13321 power law, 14

- 13322 pragmatics, 360
 13323 pre-trained word embeddings, 321
 13324 pre-trained word representations, 348
 13325 precision, 92, 476
 13326 precision-at- k , 93, 401
 13327 precision-recall curve, 420
 13328 precision-recall curves, 93
 13329 predicate, 405
 13330 predicative adjectives, 223
 13331 predictive likelihood, 113
 13332 prepositional phrase, 14, 223
 13333 presence, 90
 13334 primal form, 485
 13335 principle of compositionality, 292
 13336 prior, 476
 13337 prior expectation, 481
 13338 probabilistic context-free grammar, 225
 13339 probabilistic context-free grammars (PCFGs), 241
 13340 probabilistic models, 480
 13342 probabilistic topic model, 411
 13343 probability distribution, 478
 13344 probability mass function, 95
 13345 probability simplex, 31
 13346 processes, 424
 13347 production rules, 215
 13348 productivity, 201
 13349 projection function, 122
 13350 projectivity, 263
 13351 pronominal anaphora resolution, 359
 13352 pronoun, 186
 13353 PropBank, 312
 13354 proper nouns, 185
 13355 property (logic), 289
 13356 proposal distribution, 455
 13357 proposition, 424
 13358 propositions, 286, 287
 13359 prosody, 195
 13360 proto-roles, 311
 13361 pseudo-projective dependency parsing, 273
 13363 pumping lemma, 213
 13364 pushdown automata, 215
 13365 pushdown automaton, 256
 13366 quadratic program, 45
 13367 quasi-Newton optimization, 52
 13368 question answering, 353, 407
 13369 random outcomes, 473
 13370 ranking, 408
 13371 ranking loss, 408
 13372 recall, 92, 476
 13373 recall-at- k , 402
 13374 receiver operating characteristic (ROC), 93
 13375 rectified linear unit (ReLU), 63, 321
 13377 recurrent neural network, 145, 322
 13378 recurrent neural networks, 417
 13379 recursion, 14
 13380 recursive neural network, 399
 13381 recursive neural networks, 255, 351, 354
 13382 recursive production, 215
 13383 reference arguments, 330
 13384 reference resolution, 359
 13385 reference translations, 435
 13386 referent, 359
 13387 referring expression, 385
 13388 referring expressions, 359, 458
 13389 reflexive pronoun, 361
 13390 regression, 82
 13391 regular expression, 198
 13392 regular language, 198
 13393 regularization, 47
 13394 reification (events), 307
 13395 reinforcement learning, 453
 13396 relation extraction, 279, 328, 413
 13397 relations, 405
 13398 relations (information extraction), 405
 13399 relations (logic), 286
 13400 relative frequency estimate, 34, 136, 481
 13401 reranking, 255

- 13402 residual networks, 64
 13403 retrofitting (word embeddings), 351
 13404 Rhetorical Structure Theory (RST), 392
 13405 rhetorical zones, 385
 13406 RIBES (translation metric), 437
 13407 ridge regression, 82
 13408 risk, 454
 13409 roll-in (reinforcement learning), 374
 13410 roll-out (reinforcement learning), 375
 13411 root (morpheme), 351
- 13412 saddle point, 484
 13413 saddle points, 70
 13414 sample space, 473
 13415 satellite (RST), 393
 13416 satisfaction (logic), 291
 13417 scheduled sampling, 453
 13418 schema, 405, 406, 421
 13419 search error, 257, 372
 13420 second-order dependency parsing, 265
 13421 second-order logic, 290
 13422 seed lexicon, 83
 13423 segmented discourse representation theory (SDRT), 388
 13424 self-attention, 449
 13425 self-training, 118
 13427 semantic, 184
 13428 semantic concordance, 86
 13429 semantic parsing, 292
 13430 semantic role, 308
 13431 Semantic role labeling, 308
 13432 semantic role labeling, 414, 422
 13433 semantic underspecification, 303
 13434 semantics, 247
 13435 semi-supervised learning, 105, 115, 348
 13436 semiring algebra, 180
 13437 semiring notation, 205
 13438 semisupervised, 86
 13439 senses, 311
 13440 sentence (logic), 290
 13441 sentence compression, 465
- 13442 sentence fusion, 466
 13443 sentence summarization, 464
 13444 sentiment, 79
 13445 sentiment lexicon, 30
 13446 sequence-to-sequence, 445
 13447 shift-reduce parsing, 256
 13448 shifted positive pointwise mutual information, 346
 13449 shortest-path algorithm, 203
 13451 sigmoid, 61
 13452 simplex, 125
 13453 singular value decomposition, 71
 13454 singular value decomposition (SVD), 114
 13455 singular vectors, 72
 13456 skipgram word embeddings, 344
 13457 slack variables, 46
 13458 slot filling, 418
 13459 slots (dialogue systems), 467
 13460 smooth functions, 43
 13461 smoothing, 139
 13462 soft K -means, 107
 13463 softmax, 61, 144, 418
 13464 source domain, 120
 13465 source language, 433
 13466 spanning tree, 260
 13467 sparse matrix, 338
 13468 sparsity, 54
 13469 speech acts, 195
 13470 speech recognition, 135
 13471 split constituents, 314
 13472 spurious ambiguity, 229, 271, 276, 298
 13473 squashing function, 145
 13474 squashing functions, 63
 13475 stand-off annotations, 99
 13476 Stanford Natural Language Inference corpus, 354
 13477 statistical learning theory, 40
 13479 statistical machine translation, 434
 13480 statistical significance, 94
 13481 stem, 18
 13482 stem (morphology), 201

- 13483 stemmer, 88
 13484 step size, 51, 484
 13485 stochastic gradient descent, 43, 52
 13486 stoplist, 90
 13487 stopwords, 90
 13488 string (formal language theory), 197
 13489 string-to-tree translation, 444
 13490 strong compositionality criterion (RST),
 395
 13491 strongly equivalent grammars, 216
 13493 structure induction, 178
 13494 structured attention, 462
 13495 structured perceptron, 168
 13496 structured prediction, 28
 13497 structured support vector machine, 169
 13498 subgradient, 43, 54
 13499 subjectivity detection, 81
 13500 subordinating conjunctions, 186
 13501 subordinating discourse relations, 393
 13502 sum-product algorithm, 172
 13503 summarization, 135, 397
 13504 supersenses, 348
 13505 supervised machine learning, 30
 13506 support vector machine, 45
 13507 support vectors, 45
 13508 surface form, 209
 13509 surface realization, 457
 13510 synchronous context-free grammar, 443
 13511 synonymy, 84, 333
 13512 synset (synonym set), 84
 13513 synsets, 378
 13514 syntactic dependencies, 260
 13515 syntactic path, 315
 13516 syntax, 183, 217
 13517 tagset, 184
 13518 tanh activation function, 63
 13519 target domain, 120
 13520 target language, 433
 13521 Targeted sentiment analysis, 81
 13522 tense, 185
 13523 terminal symbols (context-free
 grammars), 215
 13524 test set, 37, 105
 13526 test statistic, 94
 13527 text classification, 27
 13528 text mining, 17
 13529 text planning, 457
 13530 thematic roles, 310
 13531 third axiom of probability, 474
 13532 third-order dependency parsing, 266
 TimeML, 423
 13534 tokenization, 86, 193
 13535 tokens, 32
 13536 topic models, 17
 13537 topic segmentation, 383
 13538 trace (syntax), 228
 13539 training set, 30, 105
 13540 transduction, 198
 13541 transfer learning, 128
 13542 transformer architecture, 449
 13543 transition features, 156
 13544 transition system, 271
 13545 transition-based parsing, 231, 256
 13546 transitive closure, 369
 13547 translation error rate (TER), 437
 13548 translation model, 136
 13549 transliteration, 451
 13550 tree-adjoining grammar, 226
 13551 tree-to-string translation, 444
 13552 tree-to-tree translation, 444
 13553 treebank, 243
 13554 trellis, 158, 211
 13555 trigrams, 38
 13556 trilexical dependencies, 252
 13557 tropical semiring, 181, 205
 13558 True negative, 91
 13559 True positive, 91
 13560 true positive, 477
 13561 true positive rate, 93
 13562 truncated singular value decomposition,
 338
 13563

- | | | | |
|-------|--|-------|--------------------------------------|
| 13564 | truth conditions, 291 | 13593 | variance (learning theory), 36 |
| 13565 | tuning set, 37, 91 | 13594 | variational autoencoder, 465 |
| 13566 | Turing test, 15 | 13595 | Vauquois Pyramid, 434 |
| 13567 | two-tailed test, 95 | 13596 | verb phrase, 219 |
| 13568 | type systems, 296 | 13597 | VerbNet, 310 |
| 13569 | type-raising, 227, 296 | 13598 | verbs, 185 |
| 13570 | types, 32 | 13599 | vertical Markovization, 248 |
| 13571 | unary closure, 234 | 13600 | Viterbi algorithm, 156 |
| 13572 | unary productions, 216 | 13601 | Viterbi variable, 158 |
| 13573 | underfit, 36 | 13602 | WARP loss, 412 |
| 13574 | underflow, 31 | 13603 | weakly equivalent grammars, 216 |
| 13575 | undergeneration, 209, 219 | 13604 | weight decay, 69 |
| 13576 | Universal Dependencies, 184, 259 | 13605 | weighted context-free grammar, 238 |
| 13577 | universal quantifier, 290 | 13606 | weighted context-free grammars, 224 |
| 13578 | unlabeled precision, 236 | 13607 | weighted finite state acceptors, 203 |
| 13579 | unlabeled recall, 236 | 13608 | wide convolution, 75 |
| 13580 | unseen word, 177 | 13609 | Wikification, 407 |
| 13581 | unsupervised, 86 | 13610 | Winograd schemas, 15 |
| 13582 | unsupervised learning, 81, 105 | 13611 | word embedding, 65 |
| 13583 | utterances, 195 | 13612 | word embeddings, 59, 146, 334, 335 |
| 13584 | validation function (semantic parsing),
302 | 13613 | word representations, 334 |
| 13585 | validity (logic), 291 | 13614 | word sense disambiguation, 83 |
| 13586 | value function, 469 | 13615 | word senses, 83 |
| 13587 | value iteration, 469 | 13616 | word tokens, 86 |
| 13588 | vanishing gradient, 63 | 13617 | WordNet, 20 |
| 13589 | vanishing gradients, 147 | 13618 | world model, 286 |
| 13590 | variable (AMR), 323 | 13619 | yield (context-free grammars), 215 |
| 13591 | variance, 97 | 13620 | zero-one loss, 42 |