

1

Natural Language Processing

2

Jacob Eisenstein

3

August 8, 2018

⁴ Contents

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

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

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

107	7	Sequence labeling	157
108	7.1	Sequence labeling as classification	157
109	7.2	Sequence labeling as structure prediction	159
110	7.3	The Viterbi algorithm	161
111	7.3.1	Example	164
112	7.3.2	Higher-order features	165
113	7.4	Hidden Markov Models	165
114	7.4.1	Estimation	167
115	7.4.2	Inference	167
116	7.5	Discriminative sequence labeling with features	169
117	7.5.1	Structured perceptron	172
118	7.5.2	Structured support vector machines	172
119	7.5.3	Conditional random fields	174
120	7.6	Neural sequence labeling	179
121	7.6.1	Recurrent neural networks	179
122	7.6.2	Character-level models	181
123	7.6.3	Convolutional Neural Networks for Sequence Labeling	182
124	7.7	*Unsupervised sequence labeling	182
125	7.7.1	Linear dynamical systems	184
126	7.7.2	Alternative unsupervised learning methods	184
127	7.7.3	Semiring Notation and the Generalized Viterbi Algorithm	184
128	8	Applications of sequence labeling	189
129	8.1	Part-of-speech tagging	189
130	8.1.1	Parts-of-Speech	190
131	8.1.2	Accurate part-of-speech tagging	194
132	8.2	Morphosyntactic Attributes	196
133	8.3	Named Entity Recognition	197
134	8.4	Tokenization	199
135	8.5	Code switching	200
136	8.6	Dialogue acts	201
137	9	Formal language theory	205
138	9.1	Regular languages	206
139	9.1.1	Finite state acceptors	207
140	9.1.2	Morphology as a regular language	208
141	9.1.3	Weighted finite state acceptors	210
142	9.1.4	Finite state transducers	215
143	9.1.5	*Learning weighted finite state automata	220
144	9.2	Context-free languages	221
145	9.2.1	Context-free grammars	222

146	9.2.2 Natural language syntax as a context-free language	225
147	9.2.3 A phrase-structure grammar for English	227
148	9.2.4 Grammatical ambiguity	232
149	9.3 *Mildly context-sensitive languages	232
150	9.3.1 Context-sensitive phenomena in natural language	233
151	9.3.2 Combinatory categorial grammar	234
152	10 Context-free parsing	239
153	10.1 Deterministic bottom-up parsing	240
154	10.1.1 Recovering the parse tree	242
155	10.1.2 Non-binary productions	242
156	10.1.3 Complexity	243
157	10.2 Ambiguity	243
158	10.2.1 Parser evaluation	244
159	10.2.2 Local solutions	245
160	10.3 Weighted Context-Free Grammars	246
161	10.3.1 Parsing with weighted context-free grammars	247
162	10.3.2 Probabilistic context-free grammars	249
163	10.3.3 *Semiring weighted context-free grammars	251
164	10.4 Learning weighted context-free grammars	251
165	10.4.1 Probabilistic context-free grammars	252
166	10.4.2 Feature-based parsing	252
167	10.4.3 *Conditional random field parsing	253
168	10.4.4 Neural context-free grammars	255
169	10.5 Grammar refinement	256
170	10.5.1 Parent annotations and other tree transformations	257
171	10.5.2 Lexicalized context-free grammars	258
172	10.5.3 *Refinement grammars	262
173	10.6 Beyond context-free parsing	263
174	10.6.1 Reranking	263
175	10.6.2 Transition-based parsing	264
176	11 Dependency parsing	269
177	11.1 Dependency grammar	269
178	11.1.1 Heads and dependents	270
179	11.1.2 Labeled dependencies	271
180	11.1.3 Dependency subtrees and constituents	272
181	11.2 Graph-based dependency parsing	274
182	11.2.1 Graph-based parsing algorithms	276
183	11.2.2 Computing scores for dependency arcs	277
184	11.2.3 Learning	279

185	11.3 Transition-based dependency parsing	280
186	11.3.1 Transition systems for dependency parsing	281
187	11.3.2 Scoring functions for transition-based parsers	285
188	11.3.3 Learning to parse	286
189	11.4 Applications	289
190	III Meaning	293
191	12 Logical semantics	295
192	12.1 Meaning and denotation	296
193	12.2 Logical representations of meaning	297
194	12.2.1 Propositional logic	297
195	12.2.2 First-order logic	298
196	12.3 Semantic parsing and the lambda calculus	302
197	12.3.1 The lambda calculus	303
198	12.3.2 Quantification	305
199	12.4 Learning semantic parsers	307
200	12.4.1 Learning from derivations	308
201	12.4.2 Learning from logical forms	310
202	12.4.3 Learning from denotations	311
203	13 Predicate-argument semantics	317
204	13.1 Semantic roles	319
205	13.1.1 VerbNet	320
206	13.1.2 Proto-roles and PropBank	321
207	13.1.3 FrameNet	322
208	13.2 Semantic role labeling	324
209	13.2.1 Semantic role labeling as classification	324
210	13.2.2 Semantic role labeling as constrained optimization	327
211	13.2.3 Neural semantic role labeling	329
212	13.3 Abstract Meaning Representation	330
213	13.3.1 AMR Parsing	333
214	13.4 Applications of Predicate-Argument Semantics	334
215	14 Distributional and distributed semantics	341
216	14.1 The distributional hypothesis	341
217	14.2 Design decisions for word representations	343
218	14.2.1 Representation	343
219	14.2.2 Context	344
220	14.2.3 Estimation	345

221	14.3 Latent semantic analysis	346
222	14.4 Brown clusters	347
223	14.5 Neural word embeddings	351
224	14.5.1 Continuous bag-of-words (CBOW)	351
225	14.5.2 Skipgrams	352
226	14.5.3 Computational complexity	352
227	14.5.4 Word embeddings as matrix factorization	354
228	14.6 Evaluating word embeddings	355
229	14.6.1 Intrinsic evaluations	355
230	14.6.2 Extrinsic evaluations	356
231	14.7 Distributed representations beyond distributional statistics	357
232	14.7.1 Word-internal structure	358
233	14.7.2 Lexical semantic resources	360
234	14.8 Distributed representations of multiword units	360
235	14.8.1 Purely distributional methods	360
236	14.8.2 Distributional-compositional hybrids	361
237	14.8.3 Supervised compositional methods	362
238	14.8.4 Hybrid distributed-symbolic representations	363
239	15 Reference Resolution	367
240	15.1 Forms of referring expressions	368
241	15.1.1 Pronouns	368
242	15.1.2 Proper Nouns	373
243	15.1.3 Nominals	374
244	15.2 Algorithms for coreference resolution	374
245	15.2.1 Mention-pair models	375
246	15.2.2 Mention-ranking models	376
247	15.2.3 Transitive closure in mention-based models	377
248	15.2.4 Entity-based models	378
249	15.3 Representations for coreference resolution	383
250	15.3.1 Features	384
251	15.3.2 Distributed representations of mentions and entities	386
252	15.4 Evaluating coreference resolution	389
253	16 Discourse	393
254	16.1 Segments	393
255	16.1.1 Topic segmentation	394
256	16.1.2 Functional segmentation	395
257	16.2 Entities and reference	395
258	16.2.1 Centering theory	396
259	16.2.2 The entity grid	397

260	16.2.3 *Formal semantics beyond the sentence level	398
261	16.3 Relations	398
262	16.3.1 Shallow discourse relations	399
263	16.3.2 Hierarchical discourse relations	402
264	16.3.3 Argumentation	406
265	16.3.4 Applications of discourse relations	407
266	IV Applications	413
267	17 Information extraction	415
268	17.1 Entities	417
269	17.1.1 Entity linking by learning to rank	418
270	17.1.2 Collective entity linking	420
271	17.1.3 *Pairwise ranking loss functions	421
272	17.2 Relations	423
273	17.2.1 Pattern-based relation extraction	424
274	17.2.2 Relation extraction as a classification task	425
275	17.2.3 Knowledge base population	428
276	17.2.4 Open information extraction	432
277	17.3 Events	433
278	17.4 Hedges, denials, and hypotheticals	434
279	17.5 Question answering and machine reading	436
280	17.5.1 Formal semantics	436
281	17.5.2 Machine reading	437
282	18 Machine translation	443
283	18.1 Machine translation as a task	443
284	18.1.1 Evaluating translations	445
285	18.1.2 Data	447
286	18.2 Statistical machine translation	448
287	18.2.1 Statistical translation modeling	449
288	18.2.2 Estimation	451
289	18.2.3 Phrase-based translation	452
290	18.2.4 *Syntax-based translation	453
291	18.3 Neural machine translation	454
292	18.3.1 Neural attention	456
293	18.3.2 *Neural machine translation without recurrence	458
294	18.3.3 Out-of-vocabulary words	459
295	18.4 Decoding	461
296	18.5 Training towards the evaluation metric	463

297	19	Text generation	467
298	19.1	Data-to-text generation	467
299	19.1.1	Latent data-to-text alignment	469
300	19.1.2	Neural data-to-text generation	470
301	19.2	Text-to-text generation	474
302	19.2.1	Neural abstractive summarization	474
303	19.2.2	Sentence fusion for multi-document summarization	476
304	19.3	Dialogue	477
305	19.3.1	Finite-state and agenda-based dialogue systems	477
306	19.3.2	Markov decision processes	478
307	19.3.3	Neural chatbots	480
308	A	Probability	483
309	A.1	Probabilities of event combinations	483
310	A.1.1	Probabilities of disjoint events	484
311	A.1.2	Law of total probability	485
312	A.2	Conditional probability and Bayes' rule	485
313	A.3	Independence	487
314	A.4	Random variables	488
315	A.5	Expectations	489
316	A.6	Modeling and estimation	490
317	B	Numerical optimization	493
318	B.1	Gradient descent	494
319	B.2	Constrained optimization	494
320	B.3	Example: Passive-aggressive online learning	495
321		Bibliography	497

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 \boldsymbol{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

$\boldsymbol{x}^{(i)}$	a column vector of feature counts for instance i , often word counts
$\boldsymbol{x}_{j:k}$	elements j through k (inclusive) of a vector \boldsymbol{x}
$[\boldsymbol{x}; \boldsymbol{y}]$	vertical concatenation of two column vectors
$[\boldsymbol{x}, \boldsymbol{y}]$	horizontal concatenation of two column vectors
\boldsymbol{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 $\boldsymbol{\theta}$
$\boldsymbol{\theta} \cdot \boldsymbol{x}^{(i)}$	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}
$\text{Diag}(\boldsymbol{x})$	a matrix with \boldsymbol{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 to
₃₃₀ computers. In the past decade, natural language processing has become embedded in our
₃₃₁ daily lives: automatic machine translation is ubiquitous on the web and in social media;
₃₃₂ text classification keeps emails from collapsing under a deluge of spam; search engines
₃₃₃ have moved beyond string matching and network analysis to a high degree of linguistic
₃₃₄ sophistication; dialog systems provide an increasingly common and effective way to get
₃₃₅ and share information.

₃₃₆ These diverse applications are based on a common set of ideas, drawing on algorithms,
₃₃₇ linguistics, logic, statistics, and more. The goal of this text is to provide a survey of
₃₃₈ 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 advises
₃₄₁ 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 language
₃₄₇ processing research bear the name “computational linguistics”, and the terms may be
₃₄₈ thought of as essentially synonymous. But while there is substantial overlap, there is an
₃₄₉ important difference in focus. In linguistics, language is the object of study. Computational
₃₅₀ 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 algo-
 353 rithms and representations for processing natural human language. The goal of natural
 354 language processing is to provide new computational capabilities around human language:
 355 for example, extracting information from texts, translating between languages, answering
 356 questions, holding a conversation, taking instructions, and so on. Fundamental linguistic
 357 insights may be crucial for accomplishing these tasks, but success is ultimately measured
 358 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 from
 361 examples. Machine learning provides an array of general techniques for tasks like converting
 362 a sequence of discrete tokens in one vocabulary to a sequence of discrete tokens in another
 363 vocabulary — a generalization of what normal people might call “translation.” Much of
 364 today’s natural language processing research can be thought of as applied machine learning.
 365 However, natural language processing has characteristics that distinguish it from many of
 366 machine learning’s other application domains.

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

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

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

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

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

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

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

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

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

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

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

²This view is shared by some, but not all, prominent researchers in artificial intelligence. Michael Jordan, a specialist in machine learning, has said that if he had a billion dollars to spend on any large research project, he would spend it on natural language processing (https://www.reddit.com/r/MachineLearning/comments/2fxifv/ama_michael_i_jordan/). On the other hand, in a public discussion about the future of artificial intelligence in February 2018, computer vision researcher Yann Lecun argued that despite its many practical applications, language is perhaps “number 300” in the priority list for artificial intelligence research, and that it would be a great achievement if AI could attain the capabilities of an orangutan, which do not include language (<http://www.abigailsee.com/2018/02/21/deep-learning-structure-and-innate-priors.html>).

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

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

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

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

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

465 1.2 Three themes in natural language processing

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

472 1.2.1 Learning and knowledge

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

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

490 the understanding and production of language. The argument for the existence of such
 491 a language faculty is based on the observation that children learn language faster and
 492 from fewer examples than would be reasonably possible, if language was learned from
 493 experience alone.³ Regardless of the cognitive validity of these arguments, it seems that
 494 linguistic structures are particularly important in scenarios where training data is limited.

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

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

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

522 1.2.2 Search and learning

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

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

525 where,

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

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

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

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

549 Learning. The learning module is responsible for finding the parameters $\boldsymbol{\theta}$. This is typ-
 550 ically (but not always) done by processing a large dataset of labeled examples,
 551 $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$. Like search, learning is also approached through the framework
 552 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.)

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 reporters

589 distinct, or should we group them into a single unit? Is the kind of interview performed by
590 a journalist the same as the kind that one undergoes when applying for a job? Ontology
591 designers face many such thorny questions, and the project of ontology design hearkens
592 back to Borges' (1993) *Celestial Emporium of Benevolent Knowledge*, which divides animals
593 into:

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

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

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

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

- 618 (1.4) The blubber served them as fuel.
- 619 (1.5) ...extracting it from the blubber of the large fish ...
- 620 (1.6) Amongst oily substances, blubber has been employed as a manure.

621 These contexts form the distributional properties of the word blubber, and they link it to
622 words which can appear in similar constructions: fat, pelts, and barnacles. This distri-
623 butional perspective makes it possible to learn about meaning from unlabeled data alone;
624 unlike relational and compositional semantics, no manual annotation or expert knowledge

625 is required. Distributional semantics is thus capable of covering a huge range of linguistic
 626 phenomena. However, it lacks precision: blubber is similar to fat in one sense, to pelts in
 627 another sense, and to barnacles in still another. The question of why all these words tend
 628 to appear in the same contexts is left unanswered.

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

635 1.3 Learning to do natural language processing

636 This text began with the notes that I use for teaching Georgia Tech’s undergraduate and
 637 graduate courses on natural language processing, CS 4650 and 7650. There are several
 638 other good resources (e.g., Manning and Schütze, 1999; Jurafsky and Martin, 2009; Smith,
 639 2011; Collins, 2013), but the goal of this text is focus on a core subset of the field, unified by
 640 the concepts of learning and search. A remarkable thing about natural language processing
 641 is that so many problems can be solved by a compact set of methods:

642 Search. Viterbi, CKY, minimum spanning tree, shift-reduce, integer linear programming,
 643 beam search.

644 Learning. Naïve Bayes, logistic regression, perceptron, expectation-maximization, matrix
 645 factorization, backpropagation, recurrent neural networks.

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

651 1.3.1 Background

652 Because natural language processing draws on many different intellectual traditions, almost
 653 everyone who approaches it feels underprepared in one way or another. Here is a summary
 654 of what is expected, and where you can learn more:

655 Mathematics and machine learning. The text assumes a background in multivariate cal-
 656 culus and linear algebra: vectors, matrices, derivatives, and partial derivatives. You
 657 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 excellent sources 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 How to use this book

The textbook is organized into four main units:

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

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

⁶<https://mml-book.github.io/>

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

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

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

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

- 715 • A course focusing on machine learning should add the chapter on unsupervised learn-
 716 ing (chapter 5). The chapters on predicate-argument semantics (chapter 13), refer-
 717 ence resolution (chapter 15), and text generation (chapter 19) are particularly influ-
 718 enced by recent machine learning innovations, including deep neural networks and
 719 learning to search.
- 720 • A course with a more linguistic orientation should add the chapters on applications of
 721 sequence labeling (chapter 8), formal language theory (chapter 9), semantics (chap-
 722 ter 12 and 13), and discourse (chapter 16).
- 723 • For a course with a more applied focus — for example, a course targeting undergrad-
 724 uates — I recommend the chapters on applications of sequence labeling (chapter 8),
 725 predicate-argument semantics (chapter 13), information extraction (chapter 17), and
 726 text generation (chapter 19).

727 Acknowledgments

728 Several of my colleagues and students read early drafts of chapters in their areas of exper-
729 tise, including Yoav Artzi, Kevin Duh, Heng Ji, Jessy Li, Brendan O'Connor, Yuval Pinter,
730 Nathan Schneider, Pamela Shapiro, Noah A. Smith, Sandeep Soni, and Luke Zettlemoyer.
731 I would also like to thank the following people for helpful discussions: Kevin Murphy,
732 Shawn Ling Ramirez, William Yang Wang, and Bonnie Webber. Several students, col-
733 leagues, friends, and family found mistakes in early drafts: Parminder Bhatia, Kimberly
734 Caras, Justin Chen, Murtaza Dhuliawala, Barbara Eisenstein, Luiz C. F. Ribeiro, Chris Gu,
735 Joshua Killingsworth, Jonathan May, Taha Merghani, Gus Monod, Raghavendra Murali,
736 Nidish Nair, Brendan O'Connor, Yuval Pinter, Nathan Schneider, Jianhao Shen, Zhewei
737 Sun, Rubin Tsui, Ashwin Cunnappakkam Vinjimir, Clay Washington, Ishan Waykul, and
738 Yuyu Zhang. Clay Washington pilot tested several of the programming exercise. Special
739 thanks to the many students in Georgia Tech's CS 4650 and 7650 who suffered through
740 early versions of the text.

741

Part I

742

Learning

₇₄₃ Chapter 2

₇₄₄ Linear text classification

₇₄₅ We'll start with the problem of text classification: given a text document, assign it a discrete
₇₄₆ label $y \in \mathcal{Y}$, where \mathcal{Y} is the set of possible labels. This problem has many applications,
₇₄₇ from spam filtering to analysis of electronic health records. Text classification is also a
₇₄₈ building block that is used throughout more complex natural language processing tasks.

₇₄₉ To perform this task, the first question is how to represent each document. A common
₇₅₀ 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
₇₅₁ 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
₇₅₂ in the vocabulary.

₇₅₃ The object \mathbf{x} is a vector, but colloquially we call it a bag of words, because it includes
₇₅₄ only information about the count of each word, and not the order in which the words
₇₅₅ appear. We have thrown out grammar, sentence boundaries, paragraphs — everything but
₇₅₆ the words. Yet the bag of words model is surprisingly effective for text classification. If you
₇₅₇ see the word freeee in an email, is it a spam email? What if you see the word Bayesian?
₇₅₈ For many labeling problems, individual words can be strong predictors.

₇₅₉ To predict a label from a bag-of-words, we can assign a score to each word in the
₇₆₀ vocabulary, measuring the compatibility with the label. In the spam filtering case, we
₇₆₁ might assign a positive score to the word freeee for the label spam, and a negative score
₇₆₂ to the word Bayesian. These scores are called weights, and they are arranged in a column
₇₆₃ vector $\boldsymbol{\theta}$.

₇₆₄ Suppose that you want a multiclass classifier, where $K \triangleq |\mathcal{Y}| > 2$. For example, we
₇₆₅ might want to classify news stories about sports, celebrities, music, and business. The goal
₇₆₆ is to predict a label \hat{y} , given the bag of words \mathbf{x} , using the weights $\boldsymbol{\theta}$. For each label $y \in \mathcal{Y}$,
₇₆₇ we compute a score $\Psi(\mathbf{x}, y)$, which is a scalar measure of the compatibility between the
₇₆₈ bag-of-words \mathbf{x} and the label y . In a linear bag-of-words classifier, this score is the vector

769 inner product between the weights θ and the output of a feature function $f(\mathbf{x}, y)$,

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

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

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

773 This function returns the count of the word freeee if the label is Spam, and it returns zero
 774 otherwise. The corresponding weight θ_j then scores the compatibility of the word freeee
 775 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,

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

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

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

776 where $\underbrace{[0; 0; \dots; 0]}_{(K-1) \times V}$ is a column vector of $(K - 1) \times V$ zeros, and the semicolon indicates
 777 vertical concatenation. This arrangement is shown in Figure 2.1; the notation may seem
 778 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} ,

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

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

779 This inner product notation gives a clean separation between the data (\mathbf{x} and y) and the
 780 parameters (θ). This notation also generalizes nicely to structured prediction, in which the
 781 space of labels \mathcal{Y} is very large, and we want to model shared substructures between labels.

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 $y = \operatorname{Sign}(\beta \cdot \mathbf{x} + a)$, 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 $y \in \mathcal{Y}$.

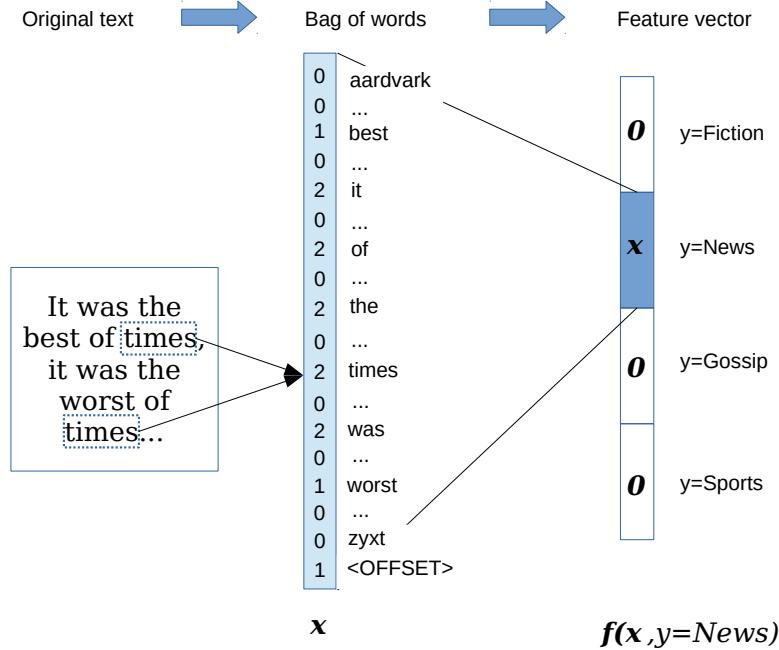


Figure 2.1: The bag-of-words and feature vector representations, for a hypothetical text classification task.

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

Returning to the weights $\boldsymbol{\theta}$, 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,

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

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

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

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

2.1 Naïve Bayes

The joint probability of a bag of words \mathbf{x} and its true label y is written $p(\mathbf{x}, y)$. Suppose we have a dataset of N labeled instances, $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$, which we assume are independent and identically distributed (IID) (see § A.3). Then the joint probability of the entire 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:

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

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

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

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

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.

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.

Algorithm 1 Generative process for the Naïve Bayes classifier

for Document $i \in \{1, 2, \dots, N\}$ do:
 Draw the label $y^{(i)} \sim \text{Categorical}(\boldsymbol{\mu})$;
 Draw the word counts $\mathbf{x}^{(i)} | y^{(i)} \sim \text{Multinomial}(\boldsymbol{\phi}_{y^{(i)}})$.

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

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

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

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 distribu-

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 .

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  $w_m^{(i)} | y^{(i)} \sim \text{Categorical}(\boldsymbol{\phi}_{y^{(i)}})$ .

```

tion is:

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

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

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

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

2.1.1 Types and tokens

A slight modification to the generative model of Naïve Bayes is shown in Algorithm 2. Instead of generating a vector of counts of types, \mathbf{x} , this model generates a sequence of tokens, $\mathbf{w} = (w_1, w_2, \dots, w_M)$. The distinction between types and tokens is critical: $x_j \in \{0, 1, 2, \dots, M\}$ is the count of word type j in the vocabulary, e.g., the number of

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

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

times the word cannibal appears; $w_m \in \mathcal{V}$ is the identity of token m in the document, e.g. $w_m = \text{cannibal}$.

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

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

2.1.2 Prediction

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

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

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

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

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

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

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

2.1.3 Estimation

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

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

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

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

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

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

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

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

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

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

883 These constraints can be incorporated by adding a set of Lagrange multipliers (see Ap-
884 pendix 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,

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

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

887 2.1.4 Smoothing and MAP estimation

888 With text data, there are likely to be pairs of labels and words that never appear in the
889 training set, leaving $\phi_{y,j} = 0$. For example, the word Bayesian may have never yet appeared
890 in a spam email. But choosing a value of $\phi_{\text{spam}, \text{Bayesian}} = 0$ would allow this single feature
891 to completely veto a label, since $p(\text{spam} | \mathbf{x}) = 0$ if $\mathbf{x}_{\text{Bayesian}} > 0$.

892 This is undesirable, because it imposes high variance: depending on what data happens
893 to be in the training set, we could get vastly different classification rules. One solution

894 is to smooth the probabilities, by adding a “pseudocount” of α to each count, and then
 895 normalizing.

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

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

898 Smoothing reduces variance, but it takes us away from the maximum likelihood es-
 899 timate: it imposes a bias. In this case, the bias points towards uniform probabilities.
 900 Machine learning theory shows that errors on heldout data can be attributed to the sum of
 901 bias and variance (Mohri et al., 2012). Techniques for reducing variance typically increase
 902 the bias, leading to a bias-variance tradeoff.

- 903 • Unbiased classifiers may overfit the training data, yielding poor performance on un-
 904 seen data.
- 905 • But if the smoothing is too large, the resulting classifier can underfit instead. In the
 906 limit of $\alpha \rightarrow \infty$, there is zero variance: you get the same classifier, regardless of the
 907 data. However, the bias is likely to be large.

908 2.1.5 Setting hyperparameters

909 How should we choose the best value of hyperparameters like α ? Maximum likelihood
 910 will not work: the maximum likelihood estimate of α on the training set will always be
 911 $\alpha = 0$. In many cases, what we really want is accuracy: the number of correct predictions,
 912 divided by the total number of predictions. (Other measures of classification performance
 913 are discussed in § 4.4.) As we will see, it is hard to optimize for accuracy directly. But for
 914 scalar hyperparameters like α can be tuned by a simple heuristic called grid search: try a
 915 set of values (e.g., $\alpha \in \{0.001, 0.01, 0.1, 1, 10\}$), compute the accuracy for each value, and
 916 choose the setting that maximizes the accuracy.

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

922 We also want to predict the performance of our classifier on unseen data. To do this,
 923 we must hold out a separate subset of data, called the test set. It is critical that the test set
 924 not overlap with either the training or development sets, or else we will overestimate the
 925 performance that the classifier will achieve on unlabeled data in the future. The test set

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.

should also not be used when making modeling decisions, such as the form of the feature function, the size of the vocabulary, and so on (these decisions are reviewed in chapter 4.) The ideal practice is to use the test set only once — otherwise, the test set is used to guide the classifier design, and test set accuracy will diverge from accuracy on truly unseen data. Because annotated data is expensive, this ideal can be hard to follow in practice, and many test sets have been used for decades. But in some high-impact applications like machine translation and information extraction, new test sets are released every year.

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

2.2 Discriminative learning

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

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

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

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

following approximation:¹¹

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

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

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

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

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

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

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

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

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

2.2.1 Perceptron

In Naïve Bayes, the weights can be interpreted as parameters of a probabilistic model. But this model requires an independence assumption that usually does not hold, and limits our choice of features. Why not forget about probability and learn the weights in an error-driven way? The perceptron algorithm, shown in Algorithm 3, is one way to do this.

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

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:      $\hat{y} \leftarrow \text{argmax}_y \boldsymbol{\theta}^{(t-1)} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y)$ 
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  $\boldsymbol{\theta}^{(t)}$ 

```

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

983 Definition 1 (Linear separability). The dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$ is linearly separable iff
 984 (if and only if) there exists some weight vector $\boldsymbol{\theta}$ and some margin ρ such that for every
 985 instance $(\mathbf{x}^{(i)}, y^{(i)})$, the inner product of $\boldsymbol{\theta}$ and the feature function for the true label,
 986 $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)})$, is at least ρ greater than inner product of $\boldsymbol{\theta}$ and the feature function for
 987 every other possible label, $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y')$.

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

988 Linear separability is important because of the following guarantee: if your data is
 989 linearly separable, then the perceptron algorithm will find a separator (Novikoff, 1962).¹²
 990 So while the perceptron may seem heuristic, it is guaranteed to succeed, if the learning
 991 problem is easy enough.

992 How useful is this proof? Minsky and Papert (1969) famously proved that the simple
 993 logical function of exclusive-or is not separable, and that a perceptron is therefore incapable
 994 of learning this function. But this is not just an issue for the perceptron: any linear
 995 classification algorithm, including Naïve Bayes, will fail on this task. In natural language

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).

classification problems usually involve high dimensional feature spaces, with thousands or millions of features. For these problems, it is very likely that the training data is indeed separable. And even if the data is not separable, it is still possible to place an upper bound on the number of errors that the perceptron algorithm will make (Freund and Schapire, 1999).

2.2.2 Averaged perceptron

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

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

Even if the data is not separable, the averaged weights will eventually converge. One possible stopping criterion is to check the difference between the average weight vectors after each pass through the data: if the norm of the difference falls below some predefined threshold, we can stop training. Another stopping criterion is to hold out some data, and to measure the predictive accuracy on this heldout data. When the accuracy on the heldout data starts to decrease, the learning algorithm has begun to overfit the training set. At this point, it is probably best to stop; this stopping criterion is known as early stopping.

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

2.3 Loss functions and large-margin classification

Naïve Bayes chooses the weights $\boldsymbol{\theta}$ by maximizing the joint log-likelihood $\log p(\mathbf{x}^{(1:N)}, y^{(1:N)})$. By convention, optimization problems are generally formulated as minimization of a loss

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:      $\hat{y} \leftarrow \text{argmax}_y \boldsymbol{\theta}^{(t-1)} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y)$ 
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  $\bar{\boldsymbol{\theta}}$ 
  
```

1032 function. The input to a loss function is the vector of weights $\boldsymbol{\theta}$, and the output is a non-
 1033 negative scalar, measuring the performance of the classifier on a training instance. The
 1034 loss $\ell(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)})$ is then a measure of the performance of the weights $\boldsymbol{\theta}$ on the instance
 1035 $(\mathbf{x}^{(i)}, y^{(i)})$. The goal of learning is to minimize the sum of the losses across all instances in
 1036 the training set.

We can trivially reformulate maximum likelihood as a loss function, by defining the loss function to be the negative log-likelihood:

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

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

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

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

1037 The problem of minimizing ℓ_{nb} is thus identical to the problem of maximum-likelihood
 1038 estimation.

1039 Loss functions provide a general framework for comparing machine learning objectives.

1040 For example, an alternative loss function is the zero-one loss,

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

1041 The zero-one loss is zero if the instance is correctly classified, and one otherwise. The sum
 1042 of zero-one losses is proportional to the error rate of the classifier on the training data. Since
 1043 a low error rate is often the ultimate goal of classification, this may seem ideal. But the
 1044 zero-one loss has several problems. One is that it is non-convex,¹³ which means that there
 1045 is no guarantee that gradient-based optimization will be effective. A more serious problem
 1046 is that the derivatives are useless: the partial derivative with respect to any parameter is
 1047 zero everywhere, except at the points where $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y) = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, \hat{y})$ for some \hat{y} . At
 1048 those points, the loss is discontinuous, and the derivative is undefined.

1049 The perceptron optimizes the following loss function:

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

1050 When $\hat{y} = y^{(i)}$, the loss is zero; otherwise, it increases linearly with the gap between the
 1051 score for the predicted label \hat{y} and the score for the true label $y^{(i)}$. Plotting this loss against
 1052 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
 1053 hinge loss.

1054 To see why this is the loss function optimized by the perceptron, take the derivative
 1055 with respect to $\boldsymbol{\theta}$,

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

1056 At each instance perceptron algorithm takes a step of magnitude one in the opposite
 1057 direction of this gradient, $\nabla_{\boldsymbol{\theta}} \ell_{\text{Perceptron}} = \frac{\partial}{\partial \boldsymbol{\theta}} \ell_{\text{Perceptron}}(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)})$. As we will see in § 2.5,
 1058 this is an example of the optimization algorithm stochastic gradient descent, applied to the
 1059 objective in Equation 2.41.

1060 Breaking ties with subgradient descent Careful readers will notice the tacit assumption
 1061 that there is a unique \hat{y} that maximizes $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y)$. What if there are two or more
 1062 labels that maximize this function? Consider binary classification: if the maximizer is $y^{(i)}$,

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.

then the gradient is zero, and so is the perceptron update; if the maximizer is $\hat{y} \neq y^{(i)}$, then the 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 loss is not smooth, because the first derivative has a discontinuity at the hinge point, where the score for the true label $y^{(i)}$ is equal to the score for some other label \hat{y} . At this point, there is no unique gradient; rather, there is a set of subgradients. A vector \mathbf{v} is a subgradient of the function g at \mathbf{u}_0 iff $g(\mathbf{u}) - g(\mathbf{u}_0) \geq \mathbf{v} \cdot (\mathbf{u} - \mathbf{u}_0)$ for all \mathbf{u} . Graphically, this defines the set of hyperplanes that include $g(\mathbf{u}_0)$ and do not intersect g at any other 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 approach from the right, the gradient is $\mathbf{0}$. At the hinge point, the subgradients include all vectors that are bounded by these two extremes. In subgradient descent, any subgradient can be used (Bertsekas, 2012). Since both $\mathbf{0}$ and $\mathbf{f}(\mathbf{x}, \hat{y}) - \mathbf{f}(\mathbf{x}, y)$ are subgradients at the hinge point, either one can be used in the perceptron update.

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

- Both ℓ_{nb} and $\ell_{\text{Perceptron}}$ are convex, making them relatively easy to optimize. However, ℓ_{nb} can be optimized in closed form, while $\ell_{\text{Perceptron}}$ requires iterating over the dataset multiple times.
- ℓ_{nb} can suffer infinite loss on a single example, since the logarithm of zero probability is negative infinity. Naïve Bayes will therefore overemphasize some examples, and underemphasize others.
- $\ell_{\text{Perceptron}}$ treats all correct answers equally. Even if $\boldsymbol{\theta}$ only gives the correct answer by a tiny margin, the loss is still zero.

2.3.1 Large margin classification

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

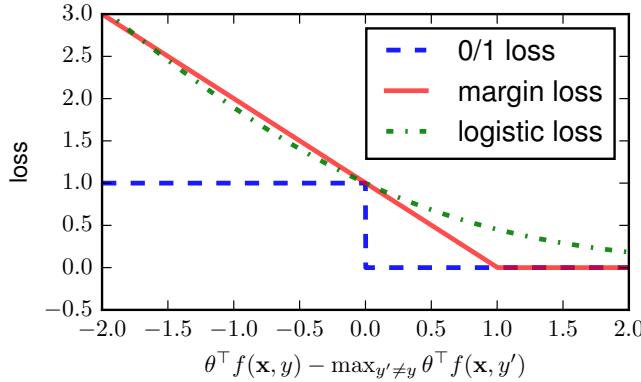


Figure 2.2: Margin, zero-one, and logistic loss functions.

loss function,

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

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

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

2.3.2 Support vector machines

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

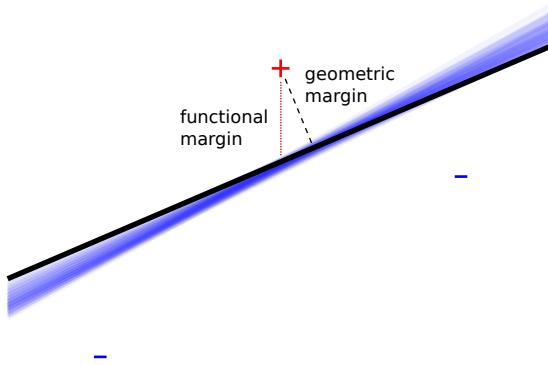


Figure 2.3: Functional and geometric margins for a binary classification problem. All separators that satisfy the margin constraint are shown. The separator with the largest geometric margin is shown in bold.

least one:

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

1095 This is a constrained optimization problem, where the second line describes constraints on
 1096 the space of possible solutions $\boldsymbol{\theta}$. In this case, the constraint is that the functional margin
 1097 always be at least one, and the objective is that the minimum geometric margin be as large
 1098 as possible.

Any scaling factor on $\boldsymbol{\theta}$ 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 $\boldsymbol{\theta}$. We therefore need only minimize the denominator $\|\boldsymbol{\theta}\|_2$, subject to the constraint on the functional margin. The minimizer of $\|\boldsymbol{\theta}\|_2$ is also the minimizer of $\frac{1}{2}\|\boldsymbol{\theta}\|_2^2 = \frac{1}{2}\sum_{j=1}^V \theta_j^2$, which is easier to work with. This gives the optimization problem,

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

1099 This optimization problem is a quadratic program: the objective is a quadratic function
 1100 of the parameters, and the constraints are all linear inequalities. The resulting classifier is

1101 better known as the support vector machine. The name derives from one of the solutions,
 1102 which is to incorporate the constraints through Lagrange multipliers $\alpha_i \geq 0, i = 1, 2, \dots, N$.
 1103 The instances for which $\alpha_i > 0$ are the support vectors; other instances are irrelevant to
 1104 the classification boundary.

1105 2.3.3 Slack variables

If a dataset is not linearly separable, then there is no $\boldsymbol{\theta}$ 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:

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

1106 The hyperparameter C controls the tradeoff between violations of the margin constraint
 1107 and the preference for a low norm of $\boldsymbol{\theta}$. As $C \rightarrow \infty$, slack is infinitely expensive, and there
 1108 is only a solution if the data is separable. As $C \rightarrow 0$, slack becomes free, and there is
 1109 a trivial solution at $\boldsymbol{\theta} = \mathbf{0}$. Thus, C plays a similar role to the smoothing parameter
 1110 in Naïve Bayes (§ 2.1.4), trading off between a close fit to the training data and better
 1111 generalization. Like the smoothing parameter of Naïve Bayes, C must be set by the user,
 1112 typically by maximizing performance on a heldout development set.

1113 To solve the constrained optimization problem defined in Equation 2.48, we can first
 1114 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,

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

1115 where each ξ_i has been substituted by the right-hand side of Equation 2.49, and the factor
 1116 of C on the slack variables has been replaced by an equivalent factor of $\lambda = \frac{1}{C}$ on the norm
 1117 of the weights.

1118 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,

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

1119 This objective maximizes over all $y \in \mathcal{Y}$, in search of labels that are both strong, as
 1120 measured by $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y)$, and wrong, as measured by $c(y^{(i)}, y)$. This maximization is known
 1121 as cost-augmented decoding, because it augments the maximization objective to favor high-
 1122 cost predictions. If the highest-scoring label is $y = y^{(i)}$, then the margin constraint is
 1123 satisfied, and the loss for this instance is zero. Cost-augmentation is only for learning: it
 1124 is not applied when making predictions on unseen data.

Differentiating Equation 2.52 with respect to the weights gives,

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

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

1125 where L_{svm} refers to minimization objective in Equation 2.52. This gradient is very similar
 1126 to the perceptron update. One difference is the additional term $\lambda \boldsymbol{\theta}$, which regularizes the
 1127 weights towards $\mathbf{0}$. The other difference is the cost $c(y^{(i)}, y)$, which is added to $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y)$
 1128 when choosing \hat{y} during training. This term derives from the margin constraint: large
 1129 margin classifiers learn not only from instances that are incorrectly classified, but also
 1130 from instances for which the correct classification decision was not sufficiently confident.

1131 2.4 Logistic regression

1132 Thus far, we have seen two broad classes of learning algorithms. Naïve Bayes is a proba-
 1133 bility method, where learning is equivalent to estimating a joint probability distribution.
 1134 The perceptron and support vector machine are discriminative, error-driven algorithms:
 1135 the learning objective is closely related to the number of errors on the training data. Prob-
 1136 abilistic and error-driven approaches each have advantages: probability makes it possible to
 1137 quantify uncertainty about the predicted labels, but the probability model of Naïve Bayes
 1138 makes unrealistic independence assumptions that limit the features that can be used.

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

Logistic regression combines advantages of discriminative and probabilistic classifiers. 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 $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y)$ 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 $\exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y))$, 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,

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

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

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

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

1139 The final line is obtained by plugging in Equation 2.55 and taking the logarithm.¹⁵ Inside
1140 the sum, we have the (additive inverse of the) logistic loss,

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

1141 The logistic loss is shown in Figure 2.2. A key difference from the zero-one and hinge losses
1142 is that logistic loss is never zero. This means that the objective function can always be
1143 improved by assigning higher confidence to the correct label.

1144 2.4.1 Regularization

1145 As with the support vector machine, better generalization can be obtained by penalizing
1146 the norm of $\boldsymbol{\theta}$. This is done by adding a term of $\frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2$ to the minimization objective.
1147 This is called L_2 regularization, because $\|\boldsymbol{\theta}\|_2^2$ is the squared L_2 norm of the vector $\boldsymbol{\theta}$.
1148 Regularization forces the estimator to trade off performance on the training data against
1149 the norm of the weights, and this can help to prevent overfitting. Consider what would
1150 happen to the unregularized weight for a base feature j that is active in only one instance

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.

1151 $\mathbf{x}^{(i)}$: the conditional log-likelihood could always be improved by increasing the weight for
 1152 this feature, so that $\theta_{(j,y^{(i)})} \rightarrow \infty$ and $\theta_{(j,\tilde{y} \neq y^{(i)})} \rightarrow -\infty$, where (j, y) is the index of feature
 1153 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,

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

1154 so that the regularization weight λ is equal to the inverse variance of the prior, $\lambda = \frac{1}{\sigma^2}$.

1155 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,

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

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

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

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

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

1156 The final step employs the definition of a conditional expectation (§ A.5). The gradient
 1157 of the logistic loss is equal to the difference between the expected counts under the current
 1158 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
 1159 vectors are equal for a single instance, there is nothing more to learn from it; when they
 1160 are equal in sum over the entire dataset, there is nothing more to learn from the dataset
 1161 as a whole. The gradient of the hinge loss is nearly identical, but it involves the features of
 1162 the predicted label under the current model, $\mathbf{f}(\mathbf{x}^{(i)}, \hat{y})$, rather than the expected features
 1163 $E_{Y|X}[\mathbf{f}(\mathbf{x}^{(i)}, y)]$ under the conditional distribution $p(y | \mathbf{x}; \boldsymbol{\theta})$.

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

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

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

1164 2.5 Optimization

1165 Each of the classification algorithms in this chapter can be viewed as an optimization
1166 problem:

- 1167 • In Naïve Bayes, the objective is the joint likelihood $\log p(\mathbf{x}^{(1:N)}, \mathbf{y}^{(1:N)})$. Maximum
1168 likelihood estimation yields a closed-form solution for $\boldsymbol{\theta}$.
- 1169 • In the support vector machine, the objective is the regularized margin loss,

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

1170 There is no closed-form solution, but the objective is convex. The perceptron algo-
1171 rithm minimizes a similar objective.

- 1172 • In logistic regression, the objective is the regularized negative log-likelihood,

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

1173 Again, there is no closed-form solution, but the objective is convex.

1174 These learning algorithms are distinguished by what is being optimized, rather than
1175 how the optimal weights are found. This decomposition is an essential feature of contem-
1176 porary machine learning. The domain expert's job is to design an objective function —
1177 or more generally, a model of the problem. If the model has certain characteristics, then
1178 generic optimization algorithms can be used to find the solution. In particular, if an ob-
1179 jective function is differentiable, then gradient-based optimization can be employed; if it
1180 is also convex, then gradient-based optimization is guaranteed to find the globally optimal
1181 solution. The support vector machine and logistic regression have both of these properties,
1182 and so are amenable to generic convex optimization techniques (Boyd and Vandenberghe,
1183 2004).

1184 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,

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

1185 where $\nabla_{\boldsymbol{\theta}} L$ is the gradient computed over the entire training set, and $\eta^{(t)}$ is the step size at
1186 iteration t . If the objective L is a convex function of $\boldsymbol{\theta}$, then this procedure is guaranteed
1187 to terminate at the global optimum, for appropriate schedule of learning rates, $\eta^{(t)}$.¹⁶

1188 In practice, gradient descent can be slow to converge, as the gradient can become
1189 infinitesimally small. Faster convergence can be obtained by second-order Newton opti-
1190 mization, which incorporates the inverse of the Hessian matrix,

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

1191 The size of the Hessian matrix is quadratic in the number of features. In the bag-of-words
1192 representation, this is usually too big to store, let alone invert. Quasi-Network optimiza-
1193 tion techniques maintain a low-rank approximation to the inverse of the Hessian matrix.
1194 Such techniques usually converge more quickly than gradient descent, while remaining
1195 computationally tractable even for large feature sets. A popular quasi-Newton algorithm
1196 is L-BFGS (Liu and Nocedal, 1989), which is implemented in many scientific computing
1197 environments, such as scipy and Matlab.

1198 For any gradient-based technique, the user must set the learning rates $\eta^{(t)}$. While con-
1199 vergence proofs usually employ a decreasing learning rate, in practice, it is common to fix
1200 $\eta^{(t)}$ to a small constant, like 10^{-3} . The specific constant can be chosen by experimentation,
1201 although there is research on determining the learning rate automatically (Schaul et al.,
1202 2013; Wu et al., 2018).

1203 2.5.2 Online optimization

1204 Batch optimization computes the objective on the entire training set before making an up-
1205 date. This may be inefficient, because at early stages of training, a small number of training

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

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

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

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  $\boldsymbol{\theta}^{(t)}$ 
```

1206 examples could point the learner in the correct direction. Online learning algorithms make
 1207 updates to the weights while iterating through the training data. The theoretical basis for
 1208 this approach is a stochastic approximation to the true objective function,

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

1209 where the instance $(\mathbf{x}^{(j)}, y^{(j)})$ is sampled at random from the full dataset.

1210 In stochastic gradient descent, the approximate gradient is computed by randomly
 1211 sampling a single instance, and an update is made immediately. This is similar to the
 1212 perceptron algorithm, which also updates the weights one instance at a time. In minibatch
 1213 stochastic gradient descent, the gradient is computed over a small set of instances. A
 1214 typical approach is to set the minibatch size so that the entire batch fits in memory on a
 1215 graphics processing unit (GPU; Neubig et al., 2017). It is then possible to speed up learning
 1216 by parallelizing the computation of the gradient over each instance in the minibatch.

1217 Algorithm 5 offers a generalized view of gradient descent. In standard gradient descent,
 1218 the batcher returns a single batch with all the instances. In stochastic gradient descent, it
 1219 returns N batches with one instance each. In mini-batch settings, the batcher returns B
 1220 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:

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

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

1221 where j iterates over features in $\mathbf{f}(\mathbf{x}, y)$.

1222 In most cases, the number of active features for any instance is much smaller than the
 1223 number of weights. If so, the computation cost of online optimization will be dominated
 1224 by the update from the regularization term, $\lambda\boldsymbol{\theta}$. The solution is to be “lazy”, updating
 1225 each θ_j only as it is used. To implement lazy updating, store an additional parameter τ_j ,
 1226 which is the iteration at which θ_j was last updated. If θ_j is needed at time t , the $t - \tau$
 1227 regularization updates can be performed all at once. This strategy is described in detail
 1228 by Kummerfeld et al. (2015).

1229 2.6 *Additional topics in classification

1230 Throughout this text, advanced topics will be marked with an asterisk.

1231 2.6.1 Feature selection by regularization

1232 In logistic regression and large-margin classification, generalization can be improved by
 1233 regularizing the weights towards 0, using the L_2 norm. But rather than encouraging
 1234 weights to be small, it might be better for the model to be sparse: it should assign weights
 1235 of exactly zero to most features, and only assign non-zero weights to features that are clearly
 1236 necessary. This idea can be formalized by the L_0 norm, $L_0 = \|\boldsymbol{\theta}\|_0 = \sum_j \delta(\theta_j \neq 0)$, which
 1237 applies a constant penalty for each non-zero weight. This norm can be thought of as a form
 1238 of feature selection: optimizing the L_0 -regularized conditional likelihood is equivalent to
 1239 trading off the log-likelihood against the number of active features. Reducing the number
 1240 of active features is desirable because the resulting model will be fast, low-memory, and
 1241 should generalize well, since irrelevant features will be pruned away. Unfortunately, the
 1242 L_0 norm is non-convex and non-differentiable. Optimization under L_0 regularization is
 1243 NP-hard, meaning that it can be solved efficiently only if P=NP (Ge et al., 2011).

1244 A useful alternative is the L_1 norm, which is equal to the sum of the absolute values of
 1245 the weights, $\|\boldsymbol{\theta}\|_1 = \sum_j |\theta_j|$. The L_1 norm is convex, and can be used as an approximation
 1246 to L_0 (Tibshirani, 1996). Conveniently, the L_1 norm also performs feature selection, by

driving many of the coefficients to zero; it is therefore known as a sparsity inducing regularizer. The L_1 norm does not have a gradient at $\theta_j = 0$, so we must instead optimize the L_1 -regularized objective using subgradient methods. The associated stochastic subgradient descent algorithms are only somewhat more complex than conventional SGD; Sra et al. (2012) survey approaches for estimation under L_1 and other regularizers.

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

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 $\boldsymbol{\theta} \cdot \mathbf{x}$. The conditional probability $p_{Y|X}$ is obtained by passing this inner product through a logistic function,

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

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

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

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

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

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

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

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

1270 where \mathcal{X} is the set of all possible feature vectors, and $p_X(\mathbf{x})$ is the probability of observing
 1271 the base features \mathbf{x} . The distribution p_X is unknown, but it can be estimated by summing
 1272 over all the instances in the training set,

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

1273 If the entropy is large, the likelihood function is smooth across possible values of y ;
 1274 if it is small, the likelihood function is sharply peaked at some preferred value; in the
 1275 limiting case, the entropy is zero if $p(y | x) = 1$ for some y . The maximum-entropy
 1276 criterion chooses to make the weakest commitments possible, while satisfying the moment-
 1277 matching constraints from Equation 2.78. The solution to this constrained optimization
 1278 problem is identical to the maximum conditional likelihood (logistic-loss) formulation that
 1279 was presented in § 2.4.

1280 2.7 Summary of learning algorithms

1281 It is natural to ask which learning algorithm is best, but the answer depends on what
 1282 characteristics are important to the problem you are trying to solve.

1283 Naïve Bayes Pros: easy to implement; estimation is fast, requiring only a single pass over
 1284 the data; assigns probabilities to predicted labels; controls overfitting with smoothing
 1285 parameter. Cons: often has poor accuracy, especially with correlated features.

1286 Perceptron Pros: easy to implement; online; error-driven learning means that accuracy
 1287 is typically high, especially after averaging. Cons: not probabilistic; hard to know
 1288 when to stop learning; lack of margin can lead to overfitting.

1289 Support vector machine Pros: optimizes an error-based metric, usually resulting in high
 1290 accuracy; overfitting is controlled by a regularization parameter. Cons: not proba-
 1291 bilistic.

1292 Logistic regression Pros: error-driven and probabilistic; overfitting is controlled by a regu-
 1293 larization parameter. Cons: batch learning requires black-box optimization; logistic
 1294 loss can “overtrain” on correctly labeled examples.

1295 One of the main distinctions is whether the learning algorithm offers a probability
 1296 over labels. This is useful in modular architectures, where the output of one classifier is
 1297 the input for some other system. In cases where probability is not necessary, the support
 1298 vector machine is usually the right choice, since it is no more difficult to implement than the
 1299 perceptron, and is often more accurate. When probability is necessary, logistic regression
 1300 is usually more accurate than Naïve Bayes.

1301 Additional resources

1302 For more on classification, you can consult a textbook on machine learning (e.g., Murphy,
 1303 2012), although the notation will differ slightly from what is typical in natural language
 1304 processing. Probabilistic methods are surveyed by Hastie et al. (2009), and Mohri et al.
 1305 (2012) emphasize theoretical considerations. Online learning is a rapidly moving subfield of
 1306 machine learning, and Bottou et al. (2016) describes progress through 2016. Kummerfeld
 1307 et al. (2015) empirically review several optimization algorithms for large-margin learning.
 1308 The python toolkit scikit-learn includes implementations of all of the algorithms described
 1309 in this chapter (Pedregosa et al., 2011).

1310 Exercises

- 1311 1. Let \mathbf{x} be a bag-of-words vector such that $\sum_{j=1}^V x_j = 1$. Verify that the multinomial
 1312 probability $p_{\text{mult}}(\mathbf{x}; \boldsymbol{\phi})$, as defined in Equation 2.12, is identical to the probability of
 1313 the same document under a categorical distribution, $p_{\text{cat}}(\mathbf{w}; \boldsymbol{\phi})$.
- 1314 2. Derive the maximum-likelihood estimate for the parameter $\boldsymbol{\mu}$ in Naïve Bayes.
- 1315 3. As noted in the discussion of averaged perceptron in § 2.2.2, the computation of the
 1316 running sum $\mathbf{m} \leftarrow \mathbf{m} + \boldsymbol{\theta}$ is unnecessarily expensive, requiring $K \times V$ operations.
 1317 Give an alternative way to compute the averaged weights $\bar{\boldsymbol{\theta}}$, with complexity that is
 1318 independent of V and linear in the sum of feature sizes $\sum_{i=1}^N |\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)})|$.
- 1319 4. Consider a dataset that is comprised of two identical instances $\mathbf{x}^{(1)} = \mathbf{x}^{(2)}$ with
 1320 distinct labels $y^{(1)} \neq y^{(2)}$. Assume all features are binary $x_j \in \{0, 1\}$ for all j .

1321 Now suppose that the averaged perceptron always chooses $i = 1$ when t is even, and
 1322 $i = 2$ when t is odd, and that it will terminate under the following condition:

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

1323 In words, the algorithm stops when the largest change in the averaged weights is less
 1324 than or equal to ϵ . Compute the number of iterations before the averaged perceptron
 1325 terminates.

- 1326 5. Suppose you have two labeled datasets D_1 and D_2 , with the same features and labels.
 - 1327 • Let $\boldsymbol{\theta}^{(1)}$ be the unregularized logistic regression (LR) coefficients from training
 1328 on dataset D_1 .
 - 1329 • Let $\boldsymbol{\theta}^{(2)}$ be the unregularized LR coefficients (same model) from training on
 1330 dataset D_2 .

- 1331 • Let θ^* be the unregularized LR coefficients from training on the combined
1332 dataset $D_1 \cup D_2$.

Under these conditions, prove that for any feature j ,

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

1333

₁₃₃₄ Chapter 3

₁₃₃₅ Nonlinear classification

₁₃₃₆ Linear classification may seem like all we need for natural language processing. The bag-
₁₃₃₇ of-words representation is inherently high dimensional, and the number of features is often
₁₃₃₈ larger than the number of training instances. This means that it is usually possible to find
₁₃₃₉ a linear classifier that perfectly fits the training data. Moving to nonlinear classification
₁₃₄₀ may therefore only increase the risk of overfitting. For many tasks, lexical features (words)
₁₃₄₁ are meaningful in isolation, and can offer independent evidence about the instance label
₁₃₄₂ — unlike computer vision, where individual pixels are rarely informative, and must be
₁₃₄₃ evaluated holistically to make sense of an image. For these reasons, natural language
₁₃₄₄ processing has historically focused on linear classification to a greater extent than other
₁₃₄₅ machine learning application domains.

₁₃₄₆ But in recent years, nonlinear classifiers have swept through natural language process-
₁₃₄₇ ing, and are now the default approach for many tasks (Manning, 2016). There are at least
₁₃₄₈ three reasons for this change.

- ₁₃₄₉ • There have been rapid advances in deep learning, a family of nonlinear methods that
₁₃₅₀ learn complex functions of the input through multiple layers of computation (Good-
₁₃₅₁ fellow et al., 2016).
- ₁₃₅₂ • Deep learning facilitates the incorporation of word embeddings, which are dense
₁₃₅₃ vector representations of words. Word embeddings can be learned from large amounts
₁₃₅₄ of unlabeled data, and enable generalization to words that do not appear in the
₁₃₅₅ annotated training data (word embeddings are discussed in detail in chapter 14).
- ₁₃₅₆ • A third reason for the rise of deep nonlinear learning algorithms is hardware. Many
₁₃₅₇ deep learning models can be implemented efficiently on graphics processing units
₁₃₅₈ (GPUs), offering substantial performance improvements over CPU-based computing.

₁₃₅₉ This chapter focuses on neural networks, which are the dominant approach for nonlinear

classification in natural language processing today.¹ Historically, a few other nonlinear learning methods have been applied to language data:

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

3.1 Feedforward neural networks

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

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

This setup is shown in Figure 3.1, which describes the proposed classifier in a computation graph: the text features \mathbf{x} are connected to the middle layer \mathbf{z} , which in turn is connected to the label y .

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

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

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

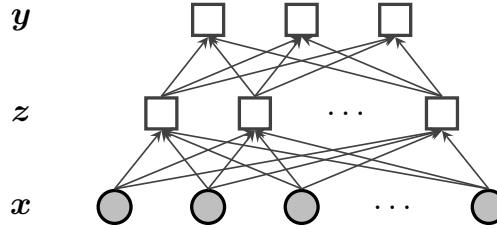


Figure 3.1: A feedforward neural network. Shaded circles indicate observed features, usually words; squares indicate nodes in the computation graph, which are computed from the information carried over the incoming arrows.

1392 where $\sigma(\cdot)$ is the sigmoid function (shown in Figure 3.2), and the matrix $\Theta^{(x \rightarrow z)} \in \mathbb{R}^{K_z \times V}$
 1393 is constructed by stacking the weight vectors for each z_k ,

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

1394 We will assume that x contains a term with a constant value of 1, so that a corresponding
 1395 offset parameter is included in each $\theta_k^{(x \rightarrow z)}$.

1396 The output layer is computed by the multi-class logistic regression probability,

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

1397 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
 1398 constructed by concatenation,

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

1399 The vector of probabilities over each possible value of y is denoted,

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

1400 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,

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

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

1401 where $\sigma(\cdot)$ is now applied elementwise to the vector of inner products,

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

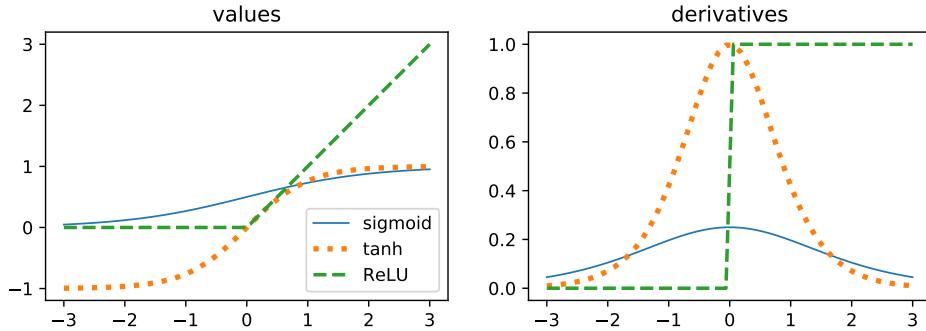


Figure 3.2: The sigmoid, tanh, and ReLU activation functions

Now suppose that the hidden features \mathbf{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 \mathbf{z} , we use the probabilities $\sigma(\boldsymbol{\theta}_k \cdot \mathbf{x})$. The resulting classifier is barely changed:

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

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

1402 This defines a classification model that predicts the label $y \in \mathcal{Y}$ from the base features \mathbf{x} ,
 1403 through a “hidden layer” \mathbf{z} . This is a feedforward neural network.²

1404 3.2 Designing neural networks

1405 This feedforward neural network can be generalized in a number of ways.

1406 3.2.1 Activation functions

1407 If the hidden layer is viewed as a set of latent features, then the sigmoid function represents
 1408 the extent to which each of these features is “activated” by a given input. However, the
 1409 hidden layer can be regarded more generally as a nonlinear transformation of the input.
 1410 This opens the door to many other activation functions, some of which are shown in
 1411 Figure 3.2. At the moment, the choice of activation functions is more art than science, but
 1412 a few points can be made about the most popular varieties:

- 1413 • The range of the sigmoid function is $(0, 1)$. The bounded range ensures that a cascade
 1414 of sigmoid functions will not “blow up” to a huge output, and this is important for

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

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 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.

1442 It is also possible to “short circuit” a hidden layer, by propagating information directly
 1443 from the input to the next higher level of the network. This is the idea behind residual
 1444 networks, which propagate information directly from the input to the subsequent layer (He
 1445 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} :

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

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

1446 where \odot refers to an elementwise vector product, and $\mathbf{1}$ is a column vector of ones. The
 1447 sigmoid function is applied elementwise to its input; recall that the output of this function
 1448 is restricted to the range $[0, 1]$. Gating is also used in the long short-term memory (LSTM),
 1449 which is discussed in chapter 6. Residual and highway connections address a problem with
 1450 deep architectures: repeated application of a nonlinear activation function can make it
 1451 difficult to learn the parameters of the lower levels of the network, which are too distant
 1452 from the supervision signal.

1453 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,

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

1454 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:

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

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

1455 where $\mathbf{e}_{y^{(i)}}$ is a one-hot vector of zeros with a value of 1 at position $y^{(i)}$. The inner product
 1456 between $\mathbf{e}_{y^{(i)}}$ and $\log \tilde{\mathbf{y}}$ is also called the multinomial cross-entropy, and this terminology
 1457 is preferred in many neural networks papers and software packages.

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

of the hidden layer is enough:

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

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

1458 In regression problems, the output is a scalar or vector (see § 4.1.2). For these problems,
 1459 a typical loss function is the squared error $(y - \hat{y})^2$ or squared norm $\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2$.

1460 3.2.4 Inputs and lookup layers

1461 In text classification, the input layer \mathbf{x} can refer to a bag-of-words vector, where x_j is the
 1462 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
 1463 represented by the vector $\boldsymbol{\theta}_j^{(x \rightarrow z)}$. This vector is sometimes described as the embedding of
 1464 word j , and can be learned from unlabeled data, using techniques discussed in chapter 14.
 1465 The columns of $\Theta^{(x \rightarrow z)}$ are each K_z -dimensional word embeddings.

1466 Chapter 2 presented an alternative view of text documents, as a sequence of word
 1467 tokens, w_1, w_2, \dots, w_M . In a neural network, each word token w_m is represented with a
 1468 one-hot vector, $\mathbf{e}_{w_m} \in \mathbb{R}^V$. The matrix-vector product $\Theta^{(x \rightarrow z)} \mathbf{e}_{w_m}$ returns the embedding
 1469 of word w_m . The complete document can be represented by horizontally concatenating these
 1470 one-hot vectors, $\mathbf{W} = [\mathbf{e}_{w_1}, \mathbf{e}_{w_2}, \dots, \mathbf{e}_{w_M}]$, and the bag-of-words representation can be
 1471 recovered from the matrix-vector product $\mathbf{W}\mathbf{1}$, which simply sums each row over the tokens
 1472 $m = \{1, 2, \dots, M\}$. The matrix product $\Theta^{(x \rightarrow z)} \mathbf{W}$ contains the horizontally concatenated
 1473 embeddings of each word in the document, which will be useful as the starting point for
 1474 convolutional neural networks (see § 3.4). This is sometimes called a lookup layer, because
 1475 the first step is to lookup the embeddings for each word in the input text.

1476 3.3 Learning neural networks

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

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

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

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

1477 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 $\boldsymbol{\Theta}^{(x \rightarrow z)}$, and $\boldsymbol{\theta}_k^{(z \rightarrow y)}$ is column k of $\boldsymbol{\Theta}^{(z \rightarrow y)}$.

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,

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

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

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

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

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

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

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 $\mathbf{x}^{(i)}$.

```

1: procedure Backprop( $G = \{f_t, \pi_t\}_{t=1}^T\}, \mathbf{x}^{(i)})$ 
2:    $v_{t(n)} \leftarrow x_n^{(i)}$  for all  $n$  and associated computation nodes  $t(n)$ .
3:   for  $t \in \text{TopologicalSort}(G)$  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  $t \in \text{Reverse}(\text{TopologicalSort}(G))$  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  $\{g_1, g_2, \dots, g_T\}$ 
```

1482 For intuition, consider each of the terms in the product.

- 1483 • 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
 1484 doesn't matter how z_k is computed, and so $\frac{\partial \ell^{(i)}}{\partial \theta_{n,k}^{(x \rightarrow z)}} \rightarrow 0$.
- 1485 • If z_k is near 1 or 0, then the curve of the sigmoid function (Figure 3.2) is nearly flat,
 1486 and changing the inputs will make little local difference. The term $z_k \times (1 - z_k)$ is
 1487 maximized at $z_k = \frac{1}{2}$, where the slope of the sigmoid function is steepest.
- 1488 • If $x_n = 0$, then it does not matter how we set the weights $\theta_{n,k}^{(x \rightarrow z)}$, so $\frac{\partial \ell^{(i)}}{\partial \theta_{n,k}^{(x \rightarrow z)}} = 0$.

1489 3.3.1 Backpropagation

1490 In the equations above, the value $\frac{\partial \ell^{(i)}}{\partial z_k}$ is reused in the derivatives with respect to each
 1491 $\theta_{n,k}^{(x \rightarrow z)}$. It should therefore be computed once, and then cached. Furthermore, we should
 1492 only compute any derivative once we have already computed all of the necessary “inputs”
 1493 demanded by the chain rule of differentiation. This combination of sequencing, caching,
 1494 and differentiation is known as backpropagation. It can be generalized to any directed
 1495 acyclic computation graph.

1496 A computation graph is a declarative representation of a computational process. At
 1497 each node t , compute a value v_t by applying a function f_t to a (possibly empty) list of
 1498 parent nodes, π_t . For example, in a feedforward network with one hidden layer, there are
 1499 nodes for the input $\mathbf{x}^{(i)}$, the hidden layer \mathbf{z} , the predicted output $\tilde{\mathbf{y}}$, and the parameters
 1500 $\{\Theta^{(x \rightarrow z)}, \Theta^{(z \rightarrow y)}, \mathbf{b}\}$. During training, there is also a node for the observed label $y^{(i)}$ and
 1501 the loss $\ell^{(i)}$. Computation graphs have three main types of nodes:

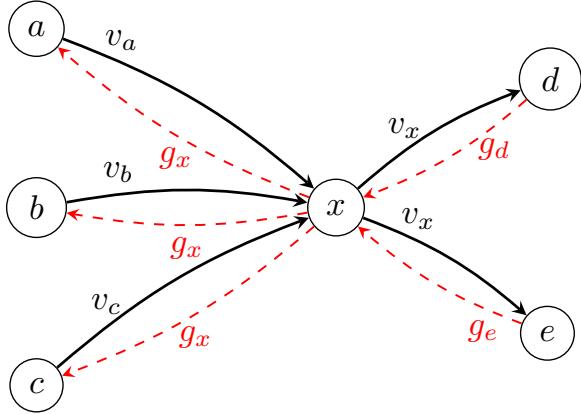


Figure 3.3: Backpropagation at a single node x in the computation graph. The values of the predecessors v_a, v_b, v_c are the inputs to x , which computes v_x , and passes it on to the successors d and e . The gradients at the successors g_d and g_e are passed back to x , where they are incorporated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

1502 Variables. The variables include the inputs \mathbf{x} , the hidden nodes \mathbf{z} , the outputs \mathbf{y} , and
 1503 the loss function. Inputs are variables that do not have parents. Backpropagation
 1504 computes the gradients with respect to all variables except the inputs, but does not
 1505 update the variables during learning.

1506 Parameters. In a feedforward network, the parameters include the weights and offsets.
 1507 Parameter nodes do not have parents, and they are updated during learning.

1508 Objective. The objective node is not the parent of any other node. Backpropagation begins
 1509 by computing the gradient with respect to this node.

1510 If the computation graph is a directed acyclic graph, then it is possible to order the
 1511 nodes with a topological sort, so that if node t is a parent of node t' , then $t < t'$. This means
 1512 that the values $\{v_t\}_{t=1}^T$ can be computed in a single forward pass. The topological sort is
 1513 reversed when computing gradients: each gradient g_t is computed from the gradients of the
 1514 children of t , implementing the chain rule of differentiation. The general backpropagation
 1515 algorithm for computation graphs is shown in Algorithm 6, and illustrated in Figure 3.3.

1516 While the gradients with respect to each parameter may be complex, they are composed
 1517 of products of simple parts. For many networks, all gradients can be computed through
 1518 automatic differentiation. This means that end users need only specify the feedforward
 1519 computation, and the gradients necessary for learning can be obtained automatically. There
 1520 are many software libraries that perform automatic differentiation on computation graphs,
 1521 such as Torch (Collobert et al., 2011), TensorFlow (Abadi et al., 2016), and DyNet (Neubig

et al., 2017). One important distinction between these libraries is whether they support dynamic computation graphs, in which the structure of the computation graph varies across instances. Static computation graphs are compiled in advance, and can be applied to fixed-dimensional data, such as bag-of-words vectors. In many natural language processing problems, each input has a distinct structure, requiring a unique computation graph.

3.3.2 Regularization and dropout

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

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

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

Another approach to controlling model complexity is dropout, which involves randomly setting some computation nodes to zero during training (Srivastava et al., 2014). For example, in the feedforward network, on each training instance, with probability ρ we set each input x_n and each hidden layer node z_k to zero. Srivastava et al. (2014) recommend $\rho = 0.5$ for hidden units, and $\rho = 0.2$ for input units. Dropout is also incorporated in the gradient computation, so if node z_k is dropped, then none of the weights $\boldsymbol{\theta}_k^{(x \rightarrow z)}$ will be updated for this instance. Dropout prevents the network from learning to depend too much on any one feature or hidden node, and prevents feature co-adaptation, in which a hidden unit is only useful in combination with one or more other hidden units. Dropout is a special case of feature noising, which can also involve adding Gaussian noise to inputs or hidden units (Holmstrom and Koistinen, 1992). Wager et al. (2013) show that dropout is approximately equivalent to “adaptive” L_2 regularization, with a separate regularization penalty for each feature.

3.3.3 *Learning theory

Chapter 2 emphasized the importance of convexity for learning: for convex objectives, the global optimum can be found efficiently. The negative log-likelihood and hinge loss are convex functions of the parameters of the output layer. However, the output of a feedforward network is generally not a convex function of the parameters of the input layer, $\boldsymbol{\Theta}^{(x \rightarrow z)}$. 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 $\mathbf{z} = [z_1, z_2, \dots, z_{K_z}]$ to $\tilde{\mathbf{z}} = [z_{\pi(1)}, z_{\pi(2)}, \dots, z_{\pi(K_z)}]$. 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 backpropagation 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 $\mathbf{x} = (0, 0)$.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 stochastic gradient descent, and by feature noising techniques such as dropout, can help online optimization to escape saddle points and find high-quality optima (Ge et al., 2015). Other techniques address saddle points directly, using local reconstructions of the Hessian matrix (Dauphin et al., 2014) or higher-order derivatives (Anandkumar and Ge, 2016).

3.3.4 Tricks

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

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

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]

1588 For the weights leading to a ReLU activation function, He et al. (2015) use similar argu-
1589 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 $\|\Theta \mathbf{x}\| = \|\mathbf{x}\|$, 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:

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

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

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

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

1590 The matrix \mathbf{U} contains the singular vectors of \mathbf{A} , and is guaranteed to be orthonormal.
1591 For more on singular value decomposition, see chapter 14.

1592 Even with careful initialization, there can still be significant variance in the final results.
1593 It can be useful to make multiple training runs, and select the one with the best performance
1594 on a heldout development set.

1595 Clipping and normalizing the gradients As already discussed, the magnitude of the gra-
1596 dient can pose problems for learning: too large, and learning can diverge, with successive
1597 updates thrashing between increasingly extreme values; too small, and learning can grind
1598 to a halt. Several heuristics have been proposed to address this issue.

- 1599 • In gradient clipping (Pascanu et al., 2013), an upper limit is placed on the norm of
 1600 the gradient, and the gradient is rescaled when this limit is exceeded,

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

- In batch normalization (Ioffe and Szegedy, 2015), the inputs to each computation 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:

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

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

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

1601 Empirically, this speeds convergence of deep architectures. One explanation is that
 1602 it helps to correct for changes in the distribution of activations during training.

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

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

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

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

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

1603 Layer normalization has similar motivations to batch normalization, but it can be
 1604 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,

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

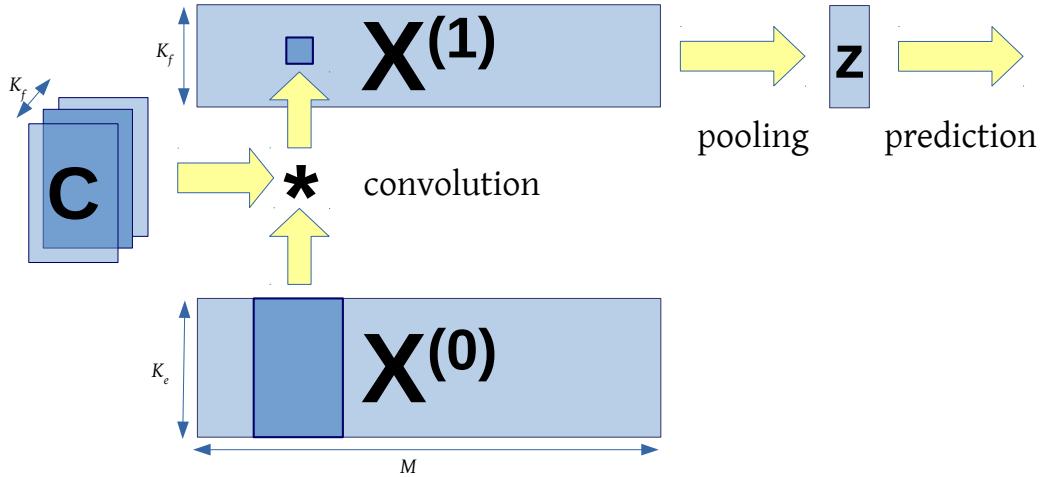


Figure 3.4: A convolutional neural network for text classification

1605 where $g_{t,j}$ is the gradient with respect to parameter j at time t , and $\beta \in [0, 1]$. This
 1606 term places more emphasis on recent gradients, and is employed in the AdaDelta (Zeiler,
 1607 2012) and Adam (Kingma and Ba, 2014) optimizers. Online optimization and its theo-
 1608 retical background are reviewed by Bottou et al. (2016). Early stopping, mentioned in
 1609 § 2.2.2, can help to avoid overfitting, by terminating training after reaching a plateau in
 1610 the performance on a heldout validation set.

1611 3.4 Convolutional neural networks

1612 A basic weakness of the bag-of-words model is its inability to account for the ways in which
 1613 words combine to create meaning, including even simple reversals such as not pleasant,
 1614 hardly a generous offer, and I wouldn't mind missing the flight. Similarly, computer vision
 1615 faces the challenge of identifying the semantics of images from pixel features that are
 1616 uninformative in isolation. An earlier generation of computer vision research focused on
 1617 designing filters to aggregate local pixel-level features into more meaningful representations,
 1618 such as edges and corners (e.g., Canny, 1987). Similarly, earlier NLP research attempted to
 1619 capture multiword linguistic phenomena by hand-designed lexical patterns (Hobbs et al.,
 1620 1997). In both cases, the output of the filters and patterns could then act as base features
 1621 in a linear classifier. But rather than designing these feature extractors by hand, a better
 1622 approach is to learn them, using the magic of backpropagation. This is the idea behind
 1623 convolutional neural networks.

1624 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,

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

1625 where f is an activation function such as tanh or ReLU, and \mathbf{b} is a vector of offsets. The
 1626 convolution operation slides the matrix $\mathbf{C}^{(k)}$ across the columns of $\mathbf{X}^{(0)}$; at each position
 1627 m , compute the elementwise product $\mathbf{C}^{(k)} \odot \mathbf{X}_{m:m+h-1}^{(0)}$, and take the sum.

1628 A simple filter might compute a weighted average over nearby words,

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

1629 thereby representing trigram units like not so unpleasant. In one-dimensional convolution,
 1630 each filter matrix $\mathbf{C}^{(k)}$ is constrained to have non-zero values only at row k (Kalchbrenner
 1631 et al., 2014).

1632 To deal with the beginning and end of the input, the base matrix $\mathbf{X}^{(0)}$ may be padded
 1633 with h column vectors of zeros at the beginning and end; this is known as wide convolution.
 1634 If padding is not applied, then the output from each layer will be $h - 1$ units smaller than
 1635 the input; this is known as narrow convolution. The filter matrices need not have identical
 1636 filter widths, so more generally we could write h_k to indicate width of filter $\mathbf{C}^{(k)}$. As
 1637 suggested by the notation $\mathbf{X}^{(0)}$, multiple layers of convolution may be applied, so that $\mathbf{X}^{(d)}$
 1638 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,

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

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

1639 The vector \mathbf{z} can now act as a layer in a feedforward network, culminating in a prediction
 1640 \hat{y} and a loss $\ell^{(i)}$. The setup is shown in Figure 3.4.

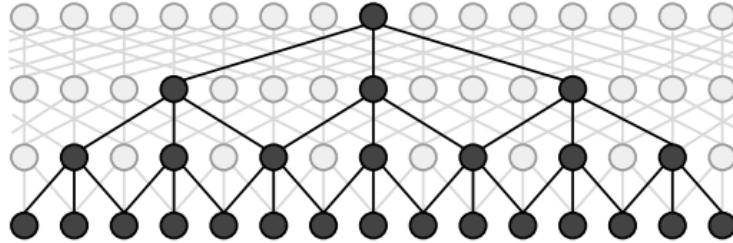


Figure 3.5: A dilated convolutional neural network captures progressively larger context through recursive application of the convolutional operator (Strubell et al., 2017) [todo: permission]

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 :

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

1641 The computer vision literature has produced a huge variety of convolutional architec-
 1642 tures, and many of these bells and whistles can be applied to text data. One avenue for
 1643 improvement is more complex pooling operations, such as k -max pooling (Kalchbrenner
 1644 et al., 2014), which returns a matrix of the k largest values for each filter. Another innova-
 1645 tion is the use of dilated convolution to build multiscale representations (Yu and Koltun,
 1646 2016). At each layer, the convolutional operator applied in strides, skipping ahead by s
 1647 steps after each feature. As we move up the hierarchy, each layer is s times smaller than
 1648 the layer below it, effectively summarizing the input. This idea is shown in Figure 3.5.
 1649 Multi-layer convolutional networks can also be augmented with “shortcut” connections, as
 1650 in the ResNet model from § 3.2.2 (Johnson and Zhang, 2017).

1651 Additional resources

1652 The deep learning textbook by Goodfellow et al. (2016) covers many of the topics in this
 1653 chapter in more detail. For a comprehensive review of neural networks in natural lan-
 1654 guage processing, see (Goldberg, 2017b). A seminal work on deep learning in natural
 1655 language processing is the aggressively titled “Natural Language Processing (Almost) from
 1656 Scratch”, which uses convolutional neural networks to perform a range of language process-
 1657 ing tasks (Collobert et al., 2011). This chapter focuses on feedforward and convolutional
 1658 neural networks, but recurrent neural networks are one of the most important deep learning

1659 architectures for natural language processing. They are covered extensively in chapters 6
 1660 and 7.

1661 The role of deep learning in natural language processing research has caused angst in
 1662 some parts of the natural language processing research community (e.g., Goldberg, 2017a),
 1663 especially as some of the more zealous deep learning advocates have argued that end-to-end
 1664 learning from “raw” text can eliminate the need for linguistic constructs such as sentences,
 1665 phrases, and even words (Zhang et al., 2015, originally titled Text understanding from
 1666 scratch). These developments were surveyed by Manning (2016).

1667 Exercises

- 1668 1. Prove that the softmax and sigmoid functions are equivalent when the number of pos-
 1669 sible labels is two. Specifically, for any $\Theta^{(z \rightarrow y)}$ (omitting the offset \mathbf{b} for simplicity),
 1670 show how to construct a vector of weights $\boldsymbol{\theta}$ such that,

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

- 1671 2. Design a feedforward network to compute the xor function:

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

1672 Your network should have a single output node which uses the Sign activation func-
 1673 tion. Use a single hidden layer, with ReLU activation functions. Describe all weights
 1674 and offsets.

- 1675 3. Consider the same network as above (with ReLU activations for the hidden layer),
 1676 with an arbitrary differentiable loss function $\ell(y^{(i)}, \tilde{y})$, where \tilde{y} is the activation of
 1677 the output node. Suppose all weights and offsets are initialized to zero. Prove that
 1678 gradient-based optimization cannot learn the desired function from this initializa-
 1679 tion.
- 1680 4. The simplest solution to the previous problem relies on the use of the ReLU activation
 1681 function at the hidden layer. Now consider a network with arbitrary activations on
 1682 the hidden layer. Show that if the initial weights are any uniform constant, then it
 1683 is not possible to learn the desired function.
- 1684 5. Consider a network in which: the base features are all binary, $\mathbf{x} \in \{0, 1\}^M$; the
 1685 hidden layer activation function is sigmoid, $z_k = \sigma(\boldsymbol{\theta}_k \cdot \mathbf{x})$; and the initial weights are
 1686 sampled independently from a standard normal distribution, $\theta_{j,k} \sim N(0, 1)$.

- 1687 • Show how the probability of a small initial gradient on any weight, $\frac{\partial z_k}{\partial \theta_{j,k}} < \alpha$,
 1688 depends on the size of the input M . Hint: use the lower bound,

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

1689 and relate this probability to the variance $V[\boldsymbol{\theta}_k \cdot \mathbf{x}]$.

- 1690 • Design an alternative initialization that removes this dependence.

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

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

1691 Define the function π as a permutation on the hidden units, as described in § 3.3.3,
 1692 so that for any Θ , $L(\Theta) = L(\Theta_\pi)$. Prove that if a feedforward network has a local
 1693 optimum in the sense of Equation 3.61, then its loss is not a convex function of the
 1694 parameters Θ , using the definition of convexity from § 2.3

- 1695 7. Consider a network with a single hidden layer, and a single output,

$$y = \boldsymbol{\theta}^{(z \rightarrow y)} \cdot g(\boldsymbol{\Theta}^{(x \rightarrow z)} \mathbf{x}). \quad [3.62]$$

1696 Assume that g is the ReLU function. Prove that for any matrix of weights $\boldsymbol{\Theta}^{(x \rightarrow z)}$, it
 1697 is permissible to rescale each row to have a norm of one, because an identical output
 1698 can be obtained by finding a corresponding rescaling of $\boldsymbol{\theta}^{(z \rightarrow y)}$.

₁₆₉₉ Chapter 4

₁₇₀₀ Linguistic applications of classification

₁₇₀₁ Having learned some techniques for classification, this chapter shifts the focus from mathematics to linguistic applications. Later in the chapter, we will consider the design decisions involved in text classification, as well as evaluation practices.

₁₇₀₄ 4.1 Sentiment and opinion analysis

₁₇₀₅ A popular application of text classification is to automatically determine the sentiment or opinion polarity of documents such as product reviews and social media posts. For example, marketers are interested to know how people respond to advertisements, services, and products (Hu and Liu, 2004); social scientists are interested in how emotions are affected by phenomena such as the weather (Hannak et al., 2012), and how both opinions and emotions spread over social networks (Coviello et al., 2014; Miller et al., 2011). In the field of digital humanities, literary scholars track plot structures through the flow of sentiment across a novel (Jockers, 2015).¹

₁₇₁₃ Sentiment analysis can be framed as a direct application of document classification, assuming reliable labels can be obtained. In the simplest case, sentiment analysis is a two or three-class problem, with sentiments of positive, negative, and possibly neutral. Such annotations could be annotated by hand, or obtained automatically through a variety of means:

- ₁₇₁₈ • Tweets containing happy emoticons can be marked as positive, sad emoticons as negative (Read, 2005; Pak and Paroubek, 2010).
- ₁₇₂₀ • Reviews with four or more stars can be marked as positive, two or fewer stars as negative (Pang et al., 2002).

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

- 1722 • Statements from politicians who are voting for a given bill are marked as positive
 1723 (towards that bill); statements from politicians voting against the bill are marked as
 1724 negative (Thomas et al., 2006).

1725 The bag-of-words model is a good fit for sentiment analysis at the document level: if
 1726 the document is long enough, we would expect the words associated with its true sentiment
 1727 to overwhelm the others. Indeed, lexicon-based sentiment analysis avoids machine learn-
 1728 ing altogether, and classifies documents by counting words against positive and negative
 1729 sentiment word lists (Taboada et al., 2011).

1730 Lexicon-based classification is less effective for short documents, such as single-sentence
 1731 reviews or social media posts. In these documents, linguistic issues like negation and
 1732 irrealis (Polanyi and Zaenen, 2006) — events that are hypothetical or otherwise non-factual
 1733 — can make bag-of-words classification ineffective. Consider the following examples:

- 1734 (4.1) That's not bad for the first day.
 1735 (4.2) This is not the worst thing that can happen.
 1736 (4.3) It would be nice if you acted like you understood.
 1737 (4.4) There is no reason at all to believe that the polluters are suddenly going to become
 1738 reasonable. (Wilson et al., 2005)
 1739 (4.5) This film should be brilliant. The actors are first grade. Stallone plays a happy,
 1740 wonderful man. His sweet wife is beautiful and adores him. He has a fascinat-
 1741 ing gift for living life fully. It sounds like a great plot, however, the film is a
 1742 failure. (Pang et al., 2002)

1743 A minimal solution is to move from a bag-of-words model to a bag-of-bigrams model,
 1744 where each base feature is a pair of adjacent words, e.g.,

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

1745 Bigrams can handle relatively straightforward cases, such as when an adjective is immedi-
 1746 ately negated; trigrams would be required to extend to larger contexts (e.g., not the worst).
 1747 But this approach will not scale to more complex examples like (4.4) and (4.5). More so-
 1748 phisticated solutions try to account for the syntactic structure of the sentence (Wilson
 1749 et al., 2005; Socher et al., 2013), or apply more complex classifiers such as convolutional
 1750 neural networks (Kim, 2014), which are described in chapter 3.

1751 4.1.1 Related problems

1752 Subjectivity Closely related to sentiment analysis is subjectivity detection, which requires
 1753 identifying the parts of a text that express subjective opinions, as well as other non-factual

1754 content such speculation and hypotheticals (Riloff and Wiebe, 2003). This can be done by
1755 treating each sentence as a separate document, and then applying a bag-of-words classifier:
1756 indeed, Pang and Lee (2004) do exactly this, using a training set consisting of (mostly)
1757 subjective sentences gathered from movie reviews, and (mostly) objective sentences gath-
1758 ered from plot descriptions. They augment this bag-of-words model with a graph-based
1759 algorithm that encourages nearby sentences to have the same subjectivity label.

1760 Stance classification In debates, each participant takes a side: for example, advocating for
1761 or against proposals like adopting a vegetarian lifestyle or mandating free college education.
1762 The problem of stance classification is to identify the author’s position from the text of the
1763 argument. In some cases, there is training data available for each position, so that standard
1764 document classification techniques can be employed. In other cases, it suffices to classify
1765 each document as whether it is in support or opposition of the argument advanced by a
1766 previous document (Anand et al., 2011). In the most challenging case, there is no labeled
1767 data for any of the stances, so the only possibility is group documents that advocate the
1768 same position (Somasundaran and Wiebe, 2009). This is a form of unsupervised learning,
1769 discussed in chapter 5.

1770 Targeted sentiment analysis The expression of sentiment is often more nuanced than a
1771 simple binary label. Consider the following examples:

1772 (4.6) The vodka was good, but the meat was rotten.

1773 (4.7) Go to Heaven for the climate, Hell for the company. –Mark Twain

1774 These statements display a mixed overall sentiment: positive towards some entities (e.g.,
1775 the vodka), negative towards others (e.g., the meat). Targeted sentiment analysis seeks to
1776 identify the writer’s sentiment towards specific entities (Jiang et al., 2011). This requires
1777 identifying the entities in the text and linking them to specific sentiment words — much
1778 more than we can do with the classification-based approaches discussed thus far. For
1779 example, Kim and Hovy (2006) analyze sentence-internal structure to determine the topic
1780 of each sentiment expression.

1781 Aspect-based opinion mining seeks to identify the sentiment of the author of a review
1782 towards predefined aspects such as price and service, or, in the case of (4.7), climate and
1783 company (Hu and Liu, 2004). If the aspects are not defined in advance, it may again be
1784 necessary to employ unsupervised learning methods to identify them (e.g., Branavan et al.,
1785 2009).

1786 Emotion classification While sentiment analysis is framed in terms of positive and nega-
1787 tive categories, psychologists generally regard emotion as more multifaceted. For example,

Ekman (1992) argues that there are six basic emotions — happiness, surprise, fear, sadness, anger, and contempt — and that they are universal across human cultures. Alm et al. (2005) build a linear classifier for recognizing the emotions expressed in children’s stories. The ultimate goal of this work was to improve text-to-speech synthesis, so that stories could be read with intonation that reflected the emotional content. They used bag-of-words features, as well as features capturing the story type (e.g., jokes, folktales), and structural features that reflect the position of each sentence in the story. The task is difficult: even human annotators frequently disagreed with each other, and the best classifiers achieved accuracy between 60-70%.

4.1.2 Alternative approaches to sentiment analysis

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

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

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

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

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

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

where \mathcal{L}_y is the lexicon for label y . Compared to the machine learning classifiers discussed in the previous chapters, lexicon-based classification may seem primitive. However, supervised machine learning relies on large annotated datasets, which are time-consuming and

1820 expensive to produce. If the goal is to distinguish two or more categories in a new domain,
1821 it may be simpler to start by writing down a list of words for each category.

1822 An early lexicon was the General Inquirer (Stone, 1966). Today, popular sentiment
1823 lexicons include sentiwordnet (Esuli and Sebastiani, 2006) and an evolving set of lexicons
1824 from Liu (2015). For emotions and more fine-grained analysis, Linguistic Inquiry and Word
1825 Count (LIWC) provides a set of lexicons (Tausczik and Pennebaker, 2010). The MPQA
1826 lexicon indicates the polarity (positive or negative) of 8221 terms, as well as whether they
1827 are strongly or weakly subjective (Wiebe et al., 2005). A comprehensive comparison of
1828 sentiment lexicons is offered by Ribeiro et al. (2016). Given an initial seed lexicon, it is
1829 possible to automatically expand the lexicon by looking for words that frequently co-occur
1830 with words in the seed set (Hatzivassiloglou and McKeown, 1997; Qiu et al., 2011).

1831 4.2 Word sense disambiguation

1832 Consider the the following headlines:

- 1833 (4.8) Iraqi head seeks arms
- 1834 (4.9) Prostitutes appeal to Pope
- 1835 (4.10) Drunk gets nine years in violin case²

1836 These headlines are ambiguous because they contain words that have multiple mean-
1837 ings, or senses. Word sense disambiguation is the problem of identifying the intended sense
1838 of each word token in a document. Word sense disambiguation is part of a larger field of
1839 research called lexical semantics, which is concerned with meanings of the words.

1840 At a basic level, the problem of word sense disambiguation is to identify the correct
1841 sense for each word token in a document. Part-of-speech ambiguity (e.g., noun versus verb)
1842 is usually considered to be a different problem, to be solved at an earlier stage. From a
1843 linguistic perspective, senses are not properties of words, but of lemmas, which are canonical
1844 forms that stand in for a set of inflected words. For example, arm/N is a lemma that
1845 includes the inflected form arms/N — the /N indicates that it we are referring to the noun,
1846 and not its homonym arm/V, which is another lemma that includes the inflected verbs
1847 (arm/V, arms/V, armed/V, arming/V). Therefore, word sense disambiguation requires first
1848 identifying the correct part-of-speech and lemma for each token, and then choosing the
1849 correct sense from the inventory associated with the corresponding lemma.³ (Part-of-
1850 speech tagging is discussed in § 8.1.)

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

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

1851 4.2.1 How many word senses?

1852 Words sometimes have many more than two senses, as exemplified by the word serve:

- 1853 • [function]: The tree stump served as a table
- 1854 • [contribute to]: His evasive replies only served to heighten suspicion
- 1855 • [provide]: We serve only the rawest fish
- 1856 • [enlist]: She served in an elite combat unit
- 1857 • [jail]: He served six years for a crime he didn't commit
- 1858 • [legal]: They were served with subpoenas⁴

1859 These sense distinctions are annotated in WordNet (<http://wordnet.princeton.edu>),
 1860 a lexical semantic database for English. WordNet consists of roughly 100,000 synsets,
 1861 which are groups of lemmas (or phrases) that are synonymous. An example synset is
 1862 {chump¹, fool², sucker¹, mark⁹}, where the superscripts index the sense of each lemma that
 1863 is included in the synset: for example, there are at least eight other senses of mark that
 1864 have different meanings, and are not part of this synset. A lemma is polysemous if it
 1865 participates in multiple synsets.

1866 WordNet defines the scope of the word sense disambiguation problem, and, more generally,
 1867 formalizes lexical semantic knowledge of English. (WordNets have been created for a
 1868 few dozen other languages, at varying levels of detail.) Some have argued that WordNet's
 1869 sense granularity is too fine (Ide and Wilks, 2006); more fundamentally, the premise that
 1870 word senses can be differentiated in a task-neutral way has been criticized as linguistically
 1871 naïve (Kilgarriff, 1997). One way of testing this question is to ask whether people tend to
 1872 agree on the appropriate sense for example sentences: according to Mihalcea et al. (2004),
 1873 people agree on roughly 70% of examples using WordNet senses; far better than chance,
 1874 but less than agreement on other tasks, such as sentiment annotation (Wilson et al., 2005).

1875 *Other lexical semantic relations Besides synonymy, WordNet also describes many other
 1876 lexical semantic relationships, including:

- 1877 • antonymy: x means the opposite of y , e.g. friend-enemy;
- 1878 • hyponymy: x is a special case of y , e.g. red-color; the inverse relationship is hyper-
- 1879 nymy;
- 1880 • meronymy: x is a part of y , e.g., wheel-bicycle; the inverse relationship is holonymy.

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

1881 Classification of these relations can be performed by searching for characteristic patterns
 1882 between pairs of words, e.g., X, such as Y, which signals hyponymy (Hearst, 1992), or X
 1883 but Y, which signals antonymy (Hatzivassiloglou and McKeown, 1997). Another approach
 1884 is to analyze each term's distributional statistics (the frequency of its neighboring words).
 1885 Such approaches are described in detail in chapter 14.

1886 4.2.2 Word sense disambiguation as classification

1887 How can we tell living plants from manufacturing plants? The context is often critical:

- 1888 (4.11) Town officials are hoping to attract new manufacturing plants through weakened
 1889 environmental regulations.
 1890 (4.12) The endangered plants play an important role in the local ecosystem.

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

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

1891 As in document classification, many of these features are irrelevant, but a few are very
 1892 strong predictors. In this example, the context word endangered is a strong signal that
 1893 the intended sense is biology rather than manufacturing. We would therefore expect
 1894 a learning algorithm to assign high weight to (endangered, biology), and low weight to
 1895 (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.,

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

1896 These are called collocation features, and they give more information about the specific role
 1897 played by each context word. This idea can be taken further by incorporating additional
 1898 syntactic information about the grammatical role played by each context feature, such as
 1899 the dependency path (see chapter 11).

1900 Using such features, a classifier can be trained from labeled data. A semantic concordance
 1901 is a corpus in which each open-class word (nouns, verbs, adjectives, and adverbs) is
 1902 tagged with its word sense from the target dictionary or thesaurus. SemCor is a semantic

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.

1903 concordance built from 234K tokens of the Brown corpus (Francis and Kucera, 1982), an-
 1904 notated as part of the WordNet project (Fellbaum, 2010). SemCor annotations look like
 1905 this:

1906 (4.13) As of Sunday_N¹ night_N¹ there was_V⁴ no word_N² ...,

1907 with the superscripts indicating the annotated sense of each polysemous word, and the
 1908 subscripts indicating the part-of-speech.

1909 As always, supervised classification is only possible if enough labeled examples can
 1910 be accumulated. This is difficult in word sense disambiguation, because each polysemous
 1911 lemma requires its own training set: having a good classifier for the senses of serve is
 1912 no help towards disambiguating plant. For this reason, unsupervised and semisupervised
 1913 methods are particularly important for word sense disambiguation (e.g., Yarowsky, 1995).
 1914 These methods will be discussed in chapter 5. Unsupervised methods typically lean on
 1915 the heuristic of “one sense per discourse”, which means that a lemma will usually have a
 1916 single, consistent sense throughout any given document (Gale et al., 1992). Based on this
 1917 heuristic, we can propagate information from high-confidence instances to lower-confidence
 1918 instances in the same document (Yarowsky, 1995).

1919 4.3 Design decisions for text classification

1920 Text classification involves a number of design decisions. In some cases, the design deci-
 1921 sion is clear from the mathematics: if you are using regularization, then a regularization
 1922 weight λ must be chosen. Other decisions are more subtle, arising only in the low level
 1923 “plumbing” code that ingests and processes the raw data. Such decision can be surprisingly
 1924 consequential for classification accuracy.

1925 4.3.1 What is a word?

1926 The bag-of-words representation presupposes that extracting a vector of word counts from
 1927 text is unambiguous. But text documents are generally represented as a sequences of
 1928 characters (in an encoding such as ascii or unicode), and the conversion to bag-of-words
 1929 presupposes a definition of the “words” that are to be counted.

1930 4.3.1.1 Tokenization

1931 The first subtask for constructing a bag-of-words vector is tokenization: converting the
 1932 text from a sequence of characters to a sequence of word tokens. A simple approach is
 1933 to define a subset of characters as whitespace, and then split the text on these tokens.
 1934 However, whitespace-based tokenization is not ideal: we may want to split conjunctions
 1935 like isn’t and hyphenated phrases like prize-winning and half-asleep, and we likely want

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? ;)

1936 to separate words from commas and periods that immediately follow them. At the same
 1937 time, it would be better not to split abbreviations like U.S. and Ph.D. In languages with
 1938 Roman scripts, tokenization is typically performed using regular expressions, with modules
 1939 designed to handle each of these cases. For example, the nltk package includes a number of
 1940 tokenizers (Loper and Bird, 2002); the outputs of four of the better-known tokenizers are
 1941 shown in Figure 4.1. Social media researchers have found that emoticons and other forms
 1942 of orthographic variation pose new challenges for tokenization, leading to the development
 1943 of special purpose tokenizers to handle these phenomena (O'Connor et al., 2010).

1944 Tokenization is a language-specific problem, and each language poses unique challenges.
 1945 For example, Chinese does not include spaces between words, nor any other consistent
 1946 orthographic markers of word boundaries. A “greedy” approach is to scan the input for
 1947 character substrings that are in a predefined lexicon. However, Xue et al. (2003) notes that
 1948 this can be ambiguous, since many character sequences could be segmented in multiple
 1949 ways. Instead, he trains a classifier to determine whether each Chinese character, or hanzi,
 1950 is a word boundary. More advanced sequence labeling methods for word segmentation are
 1951 discussed in § 8.4. Similar problems can occur in languages with alphabetic scripts, such as
 1952 German, which does not include whitespace in compound nouns, yielding examples such as
 1953 Freundschaftsbezeigungen (demonstration of friendship) and Dilettantenaufdringlichkeiten
 1954 (the importunities of dilettantes). As Twain (1997) argues, “These things are not words,
 1955 they are alphabetic processions.” Social media raises similar problems for English and
 1956 other languages, with hashtags such as #TrueLoveInFourWords requiring decomposition
 1957 for analysis (Brun and Roux, 2014).

1958 4.3.1.2 Normalization

1959 After splitting the text into tokens, the next question is which tokens are really distinct. Is
 1960 it necessary to distinguish great, Great, and GREAT? Sentence-initial capitalization may
 1961 be irrelevant to the classification task. Going further, the complete elimination of case
 1962 distinctions will result in a smaller vocabulary, and thus smaller feature vectors. However,
 1963 case distinctions might be relevant in some situations: for example, apple is a delicious
 1964 pie filling, while Apple is a company that specializes in proprietary dongles and power
 1965 adapters.

1966 For Roman script, case conversion can be performed using unicode string libraries.

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

1967 Many scripts do not have case distinctions (e.g., the Devanagari script used for South
 1968 Asian languages, the Thai alphabet, and Japanese kana), and case conversion for all scripts
 1969 may not be available in every programming environment. (Unicode support is an important
 1970 distinction between Python’s versions 2 and 3, and is a good reason for migrating to Python
 1971 3 if you have not already done so. Compare the output of the code ”\à l\’hôtel”.upper()
 1972 in the two language versions.)⁶

1973 Case conversion is a type of normalization, which refers to string transformations that
 1974 remove distinctions that are irrelevant to downstream applications (Sproat et al., 2001).
 1975 Other normalizations include the standardization of numbers (e.g., 1,000 to 1000) and
 1976 dates (e.g., August 11, 2015 to 2015/11/08). Depending on the application, it may even
 1977 be worthwhile to convert all numbers and dates to special tokens, !NUM and !DATE. In
 1978 social media, there are additional orthographic phenomena that may be normalized, such
 1979 as expressive lengthening, e.g., coooooool (Aw et al., 2006; Yang and Eisenstein, 2013).
 1980 Similarly, historical texts feature spelling variations that may need to be normalized to a
 1981 contemporary standard form (Baron and Rayson, 2008).

1982 A more extreme form of normalization is to eliminate inflectional affixes, such as the
 1983 -ed and -s suffixes in English. On this view, bike, bikes, biking, and biked all refer to
 1984 the same underlying concept, so they should be grouped into a single feature. A stemmer
 1985 is a program for eliminating affixes, usually by applying a series of regular expression
 1986 substitutions. Character-based stemming algorithms are necessarily approximate, as shown
 1987 in Figure 4.2: the Lancaster stemmer incorrectly identifies -ers as an inflectional suffix of
 1988 sisters (by analogy to fix/fixers), and both stemmers incorrectly identify -s as a suffix
 1989 of this and Williams. Fortunately, even inaccurate stemming can improve bag-of-words
 1990 classification models, by merging related strings and thereby reducing the vocabulary size.

1991 Accurately handling irregular orthography requires word-specific rules. Lemmatizers
 1992 are systems that identify the underlying lemma of a given wordform. They must avoid the
 1993 over-generalization errors of the stemmers in Figure 4.2, and also handle more complex
 1994 transformations, such as geese→goose. The output of the WordNet lemmatizer is shown
 1995 in the final line of Figure 4.2. Both stemming and lemmatization are language-specific: an

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

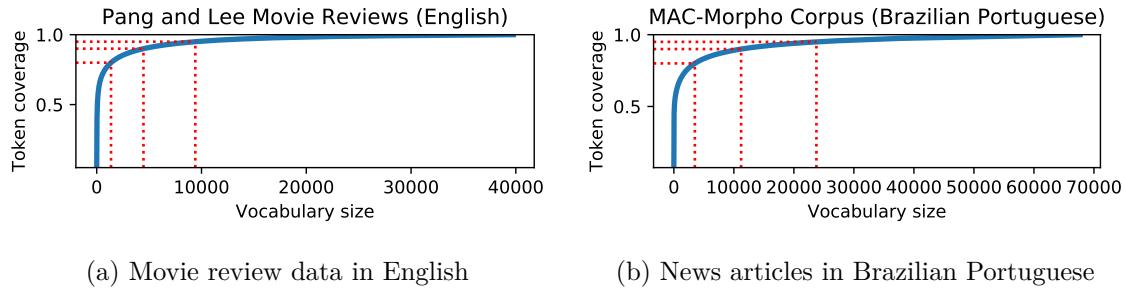


Figure 4.3: Tradeoff between token coverage (y-axis) and vocabulary size, on the nltk movie review dataset, after sorting the vocabulary by decreasing frequency. The red dashed lines indicate 80%, 90%, and 95% coverage.

1996 English stemmer or lemmatizer is of little use on a text written in another language. The
 1997 discipline of morphology relates to the study of word-internal structure, and is described
 1998 in more detail in § 9.1.2.

1999 The value of normalization depends on the data and the task. Normalization reduces
 2000 the size of the feature space, which can help in generalization. However, there is always the
 2001 risk of merging away linguistically meaningful distinctions. In supervised machine learning,
 2002 regularization and smoothing can play a similar role to normalization — preventing the
 2003 learner from overfitting to rare features — while avoiding the language-specific engineering
 2004 required for accurate normalization. In unsupervised scenarios, such as content-based
 2005 information retrieval (Manning et al., 2008) and topic modeling (Blei et al., 2003), normalization
 2006 is more critical.

2007 4.3.2 How many words?

2008 Limiting the size of the feature vector reduces the memory footprint of the resulting models,
 2009 and increases the speed of prediction. Normalization can help to play this role, but a more
 2010 direct approach is simply to limit the vocabulary to the N most frequent words in the
 2011 dataset. For example, in the movie-reviews dataset provided with nltk (originally from
 2012 Pang et al., 2002), there are 39,768 word types, and 1.58M tokens. As shown in Figure 4.3a,
 2013 the most frequent 4000 word types cover 90% of all tokens, offering an order-of-magnitude
 2014 reduction in the model size. Such ratios are language-specific: in for example, in the
 2015 Brazilian Portuguese Mac-Morpho corpus (Aluísio et al., 2003), attaining 90% coverage
 2016 requires more than 10000 word types (Figure 4.3b). This reflects the morphological
 2017 complexity of Portuguese, which includes many more inflectional suffixes than English.

2018 Eliminating rare words is not always advantageous for classification performance: for
 2019 example, names, which are typically rare, play a large role in distinguishing topics of news
 2020 articles. Another way to reduce the size of the feature space is to eliminate stopwords such

as the, to, and and, which may seem to play little role in expressing the topic, sentiment, or stance. This is typically done by creating a stoplist (e.g., `nltk.corpus.stopwords`), and then ignoring all terms that match the list. However, corpus linguists and social psychologists have shown that seemingly inconsequential words can offer surprising insights about the author or nature of the text (Biber, 1991; Chung and Pennebaker, 2007). Furthermore, high-frequency words are unlikely to cause overfitting in discriminative classifiers. As with normalization, stopword filtering is more important for unsupervised problems, such as term-based document retrieval.

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

4.3.3 Count or binary?

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

4.4 Evaluating classifiers

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

There are a number of ways to evaluate classifier performance. The simplest is accuracy:

2054 the number of correct predictions, divided by the total number of instances,

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

2055 Exams are usually graded by accuracy. Why are other metrics necessary? The main
 2056 reason is class imbalance. Suppose you are building a classifier to detect whether an
 2057 electronic health record (EHR) describes symptoms of a rare disease, which appears in
 2058 only 1% of all documents in the dataset. A classifier that reports $\hat{y} = \text{Negative}$ for all
 2059 documents would achieve 99% accuracy, but would be practically useless. We need metrics
 2060 that are capable of detecting the classifier's ability to discriminate between classes, even
 2061 when the distribution is skewed.

2062 One solution is to build a balanced test set, in which each possible label is equally
 2063 represented. But in the EHR example, this would mean throwing away 98% of the original
 2064 dataset! Furthermore, the detection threshold itself might be a design consideration: in
 2065 health-related applications, we might prefer a very sensitive classifier, which returned a
 2066 positive prediction if there is even a small chance that $y^{(i)} = \text{Positive}$. In other applications,
 2067 a positive result might trigger a costly action, so we would prefer a classifier that only makes
 2068 positive predictions when absolutely certain. We need additional metrics to capture these
 2069 characteristics.

2070 4.4.1 Precision, recall, and F -measure

2071 For any label (e.g., positive for presence of symptoms of a disease), there are two possible
 2072 errors:

- 2073 • False positive: the system incorrectly predicts the label.
- 2074 • False negative: the system incorrectly fails to predict the label.

2075 Similarly, for any label, there are two ways to be correct:

- 2076 • True positive: the system correctly predicts the label.
- 2077 • True negative: the system correctly predicts that the label does not apply to this
 2078 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:

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

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

2079 Recall and precision are both conditional likelihoods of a correct prediction, which is why
 2080 their numerators are the same. Recall is conditioned on k being the correct label, $y^{(i)} = k$,
 2081 so the denominator sums over true positive and false negatives. Precision is conditioned
 2082 on k being the prediction, so the denominator sums over true positives and false positives.
 2083 Note that true negatives are not considered in either statistic. The classifier that labels
 2084 every document as “negative” would achieve zero recall; precision would be $\frac{0}{0}$.

2085 Recall and precision are complementary. A high-recall classifier is preferred when false
 2086 negatives are cheaper than false positives: for example, in a preliminary screening for
 2087 symptoms of a disease, the cost of a false positive might be an additional test, while a false
 2088 negative would result in the disease going untreated. Conversely, a high-precision classifier
 2089 is preferred when false positives are more expensive: for example, in spam detection, a
 2090 false negative is a relatively minor inconvenience, while a false positive might mean that
 2091 an important message goes unread.

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

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

2092 where r is recall and p is precision.⁷

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

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

2093 In multi-class problems with unbalanced class distributions, the macro F -measure is a
 2094 balanced measure of how well the classifier recognizes each class. In micro F -measure, we
 2095 compute true positives, false positives, and false negatives for each class, and then add
 2096 them up to compute a single recall, precision, and F -measure. This metric is balanced
 2097 across instances rather than classes, so it weights each class in proportion to its frequency
 2098 — 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.

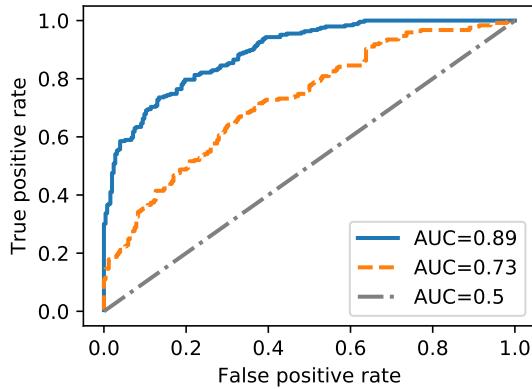


Figure 4.4: ROC curves for three classifiers of varying discriminative power, measured by AUC (area under the curve)

2099 4.4.2 Threshold-free metrics

2100 In binary classification problems, it is possible to trade off between recall and precision by
 2101 adding a constant “threshold” to the output of the scoring function. This makes it possible
 2102 to trace out a curve, where each point indicates the performance at a single threshold. In
 2103 the receiver operating characteristic (ROC) curve,⁸ the x -axis indicates the false positive
 2104 rate, $\frac{FP}{FP+TN}$, and the y-axis indicates the recall, or true positive rate. A perfect classifier
 2105 attains perfect recall without any false positives, tracing a “curve” from the origin (0,0) to
 2106 the upper left corner (0,1), and then to (1,1). In expectation, a non-discriminative classifier
 2107 traces a diagonal line from the origin (0,0) to the upper right corner (1,1). Real classifiers
 2108 tend to fall between these two extremes. Examples are shown in Figure 4.4.

2109 The ROC curve can be summarized in a single number by taking its integral, the area
 2110 under the curve (auc). The auc can be interpreted as the probability that a randomly-selected
 2111 positive example will be assigned a higher score by the classifier than a randomly-selected
 2112 negative example. A perfect classifier has auc = 1 (all positive examples score higher than
 2113 all negative examples); a non-discriminative classifier has auc = 0.5 (given a randomly
 2114 selected positive and negative example, either could score higher with equal probability);
 2115 a perfectly wrong classifier would have auc = 0 (all negative examples score higher than
 2116 all positive examples). One advantage of AUC in comparison to F -measure is that the
 2117 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.

2118 4.4.3 Classifier comparison and statistical significance

2119 Natural language processing research and engineering often involves comparing different
 2120 classification techniques. In some cases, the comparison is between algorithms, such as
 2121 logistic regression versus averaged perceptron, or L_2 regularization versus L_1 . In other
 2122 cases, the comparison is between feature sets, such as the bag-of-words versus positional
 2123 bag-of-words (see § 4.2.2). Ablation testing involves systematically removing (ablating)
 2124 various aspects of the classifier, such as feature groups, and testing the null hypothesis
 2125 that the ablated classifier is as good as the full model.

2126 A full treatment of hypothesis testing is beyond the scope of this text, but this section
 2127 contains a brief summary of the techniques necessary to compare classifiers. The main
 2128 aim of hypothesis testing is to determine whether the difference between two statistics —
 2129 for example, the accuracies of two classifiers — is likely to arise by chance. We will be
 2130 concerned with chance fluctuations that arise due to the finite size of the test set.⁹ An
 2131 improvement of 10% on a test set with ten instances may reflect a random fluctuation
 2132 that makes the test set more favorable to classifier c_1 than c_2 ; on another test set with
 2133 a different ten instances, we might find that c_2 does better than c_1 . But if we observe
 2134 the same 10% improvement on a test set with 1000 instances, this is highly unlikely to
 2135 be explained by chance. Such a finding is said to be statistically significant at a level p ,
 2136 which is the probability of observing an effect of equal or greater magnitude when the null
 2137 hypothesis is true. The notation $p < .05$ indicates that the likelihood of an equal or greater
 2138 effect is less than 5%, assuming the null hypothesis is true.¹⁰

2139 4.4.3.1 The binomial test

2140 The statistical significance of a difference in accuracy can be evaluated using classical tests,
 2141 such as the binomial test.¹¹ Suppose that classifiers c_1 and c_2 disagree on N instances in
 2142 a test set with binary labels, and that c_1 is correct on k of those instances. Under the null
 2143 hypothesis that the classifiers are equally accurate, we would expect k/N to be roughly
 2144 equal to 1/2, and as N increases, k/N should be increasingly close to this expected value.
 2145 These properties are captured by the binomial distribution, which is a probability over

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.

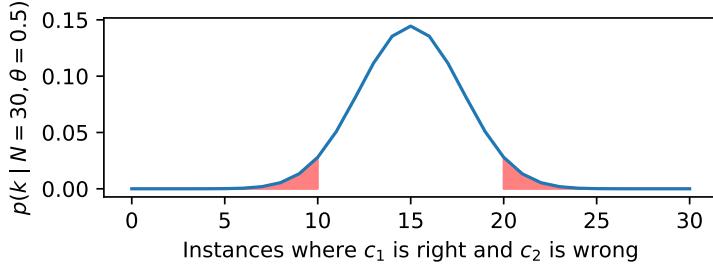


Figure 4.5: Probability mass function for the binomial distribution. The pink highlighted areas represent the cumulative probability for a significance test on an observation of $k = 10$ and $N = 30$.

counts 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 accurate,

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

2160 we would take the sum of one-tailed p -values, where the second term is computed from the
 2161 right tail of Figure 4.5. The binomial distribution is symmetric, so this can be computed
 2162 by simply doubling the one-tailed p -value.

2163 Two-tailed tests are more stringent, but they are necessary in cases in which there is
 2164 no prior intuition about whether c_1 or c_2 is better. For example, in comparing logistic
 2165 regression versus averaged perceptron, a two-tailed test is appropriate. In an ablation test,
 2166 c_2 may contain a superset of the features available to c_1 . If the additional features are
 2167 thought to be likely to improve performance, then a one-tailed test would be appropriate,
 2168 if chosen in advance. However, such a test can only prove that c_2 is more accurate than
 2169 c_1 , and not the reverse.

2170 4.4.3.2 *Randomized testing

2171 The binomial test is appropriate for accuracy, but not for more complex metrics such
 2172 as F -measure. To compute statistical significance for arbitrary metrics, we can apply
 2173 randomization. Specifically, draw a set of M bootstrap samples (Efron and Tibshirani,
 2174 1993), by resampling instances from the original test set with replacement. Each bootstrap
 2175 sample is itself a test set of size N . Some instances from the original test set will not appear
 2176 in any given bootstrap sample, while others will appear multiple times; but overall, the
 2177 sample will be drawn from the same distribution as the original test set. We can then
 2178 compute any desired evaluation on each bootstrap sample, which gives a distribution over
 2179 the value of the metric. Algorithm 7 shows how to perform this computation.

2180 To compare the F -measure of two classifiers c_1 and c_2 , we set the function $\delta(\cdot)$ to
 2181 compute the difference in F -measure on the bootstrap sample. If the difference is less than
 2182 or equal to zero in at least 5% of the samples, then we cannot reject the one-tailed null
 2183 hypothesis that c_2 is at least as good as c_1 (Berg-Kirkpatrick et al., 2012). We may also be
 2184 interested in the 95% confidence interval around a metric of interest, such as the F -measure
 2185 of a single classifier. This can be computed by sorting the output of Algorithm 7, and then

2186 setting the top and bottom of the 95% confidence interval to the values at the 2.5% and
2187 97.5% percentiles of the sorted outputs. Alternatively, you can fit a normal distribution to
2188 the set of differences across bootstrap samples, and compute a Gaussian confidence interval
2189 from the mean and variance.

2190 As the number of bootstrap samples goes to infinity, $M \rightarrow \infty$, the bootstrap estimate
2191 is increasingly accurate. A typical choice for M is 10^4 or 10^5 ; larger numbers of samples
2192 are necessary for smaller p -values. One way to validate your choice of M is to run the test
2193 multiple times, and ensure that the p -values are similar; if not, increase M by an order
2194 of magnitude. This is a heuristic measure of the variance of the test, which can decrease
2195 with the square root \sqrt{M} (Robert and Casella, 2013).

2196 4.4.4 *Multiple comparisons

2197 Sometimes it is necessary to perform multiple hypothesis tests, such as when comparing
2198 the performance of several classifiers on multiple datasets. Suppose you have five datasets,
2199 and you compare four versions of your classifier against a baseline system, for a total of
2200 20 comparisons. Even if none of your classifiers is better than the baseline, there will
2201 be some chance variation in the results, and in expectation you will get one statistically
2202 significant improvement at $p = 0.05 = \frac{1}{20}$. It is therefore necessary to adjust the p -values
2203 when reporting the results of multiple comparisons.

2204 One approach is to require a threshold of $\frac{\alpha}{m}$ to report a p value of $p < \alpha$ when performing
2205 m tests. This is known as the Bonferroni correction, and it limits the overall probability
2206 of incorrectly rejecting the null hypothesis at α . Another approach is to bound the false
2207 discovery rate (FDR), which is the fraction of null hypothesis rejections that are incorrect.
2208 Benjamini and Hochberg (1995) propose a p -value correction that bounds the fraction of
2209 false discoveries at α : sort the p -values of each individual test in ascending order, and
2210 set the significance threshold equal to largest k such that $p_k \leq \frac{k}{m}\alpha$. If $k > 1$, the FDR
2211 adjustment is more permissive than the Bonferroni correction.

2212 4.5 Building datasets

2213 Sometimes, if you want to build a classifier, you must first build a dataset of your own. This
2214 includes selecting a set of documents or instances to annotate, and then performing the
2215 annotations. The scope of the dataset may be determined by the application: if you want
2216 to build a system to classify electronic health records, then you must work with a corpus
2217 of records of the type that your classifier will encounter when deployed. In other cases,
2218 the goal is to build a system that will work across a broad range of documents. In this
2219 case, it is best to have a balanced corpus, with contributions from many styles and genres.
2220 For example, the Brown corpus draws from texts ranging from government documents to
2221 romance novels (Francis, 1964), and the Google Web Treebank includes annotations for

2222 five “domains” of web documents: question answers, emails, newsgroups, reviews, and
2223 blogs (Petrov and McDonald, 2012).

2224 4.5.1 Metadata as labels

2225 Annotation is difficult and time-consuming, and most people would rather avoid it. It is
2226 sometimes possible to exploit existing metadata to obtain labels for training a classifier.
2227 For example, reviews are often accompanied by a numerical rating, which can be converted
2228 into a classification label (see § 4.1). Similarly, the nationalities of social media users
2229 can be estimated from their profiles (Dredze et al., 2013) or even the time zones of their
2230 posts (Gouws et al., 2011). More ambitiously, we may try to classify the political affiliations
2231 of social media profiles based on their social network connections to politicians and major
2232 political parties (Rao et al., 2010).

2233 The convenience of quickly constructing large labeled datasets without manual annotation
2234 is appealing. However this approach relies on the assumption that unlabeled instances —
2235 for which metadata is unavailable — will be similar to labeled instances. Consider the
2236 example of labeling the political affiliation of social media users based on their network
2237 ties to politicians. If a classifier attains high accuracy on such a test set, is it safe to assume
2238 that it accurately predicts the political affiliation of all social media users? Probably not.
2239 Social media users who establish social network ties to politicians may be more likely to
2240 mention politics in the text of their messages, as compared to the average user, for whom
2241 no political metadata is available. If so, the accuracy on a test set constructed from social
2242 network metadata would give an overly optimistic picture of the method’s true performance
2243 on unlabeled data.

2244 4.5.2 Labeling data

2245 In many cases, there is no way to get ground truth labels other than manual annotation.
2246 An annotation protocol should satisfy several criteria: the annotations should be expressive
2247 enough to capture the phenomenon of interest; they should be replicable, meaning that
2248 another annotator or team of annotators would produce very similar annotations if given
2249 the same data; and they should be scalable, so that they can be produced relatively quickly.
2250 Hovy and Lavid (2010) propose a structured procedure for obtaining annotations that meet
2251 these criteria, which is summarized below.

- 2252 1. Determine what the annotations are to include. This is usually based on some theory
2253 of the underlying phenomenon: for example, if the goal is to produce annotations
2254 about the emotional state of a document’s author, one should start with a theoretical
2255 account of the types or dimensions of emotion (e.g., Mohammad and Turney, 2013).
2256 At this stage, the tradeoff between expressiveness and scalability should be considered:

2257 a full instantiation of the underlying theory might be too costly to annotate at scale,
2258 so reasonable approximations should be considered.

- 2259 2. Optionally, one may design or select a software tool to support the annotation effort.
2260 Existing general-purpose annotation tools include BRAT (Stenetorp et al., 2012) and
2261 MMAX2 (Müller and Strube, 2006).
- 2262 3. Formalize the instructions for the annotation task. To the extent that the instructions
2263 are not explicit, the resulting annotations will depend on the intuitions of the annotators.
2264 These intuitions may not be shared by other annotators, or by the users of the
2265 annotated data. Therefore explicit instructions are critical to ensuring the annotations
2266 are replicable and usable by other researchers.
- 2267 4. Perform a pilot annotation of a small subset of data, with multiple annotators for
2268 each instance. This will give a preliminary assessment of both the replicability and
2269 scalability of the current annotation instructions. Metrics for computing the rate
2270 of agreement are described below. Manual analysis of specific disagreements should
2271 help to clarify the instructions, and may lead to modifications of the annotation task
2272 itself. For example, if two labels are commonly conflated by annotators, it may be
2273 best to merge them.
- 2274 5. Annotate the data. After finalizing the annotation protocol and instructions, the
2275 main annotation effort can begin. Some, if not all, of the instances should receive
2276 multiple annotations, so that inter-annotator agreement can be computed. In some
2277 annotation projects, instances receive many annotations, which are then aggregated
2278 into a “consensus” label (e.g., Danescu-Niculescu-Mizil et al., 2013). However, if the
2279 annotations are time-consuming or require significant expertise, it may be preferable
2280 to maximize scalability by obtaining multiple annotations for only a small subset of
2281 examples.
- 2282 6. Compute and report inter-annotator agreement, and release the data. In some cases,
2283 the raw text data cannot be released, due to concerns related to copyright or privacy.
2284 In these cases, one solution is to publicly release stand-off annotations, which contain
2285 links to document identifiers. The documents themselves can be released under the
2286 terms of a licensing agreement, which can impose conditions on how the data is
2287 used. It is important to think through the potential consequences of releasing data:
2288 people may make personal data publicly available without realizing that it could be
2289 redistributed in a dataset and publicized far beyond their expectations (boyd and
2290 Crawford, 2012).

2291 4.5.2.1 Measuring inter-annotator agreement

2292 To measure the replicability of annotations, a standard practice is to compute the extent
 2293 to which annotators agree with each other. If the annotators frequently disagree, this casts
 2294 doubt on either their reliability or on the annotation system itself. For classification, one
 2295 can compute the frequency with which the annotators agree; for rating scales, one can
 2296 compute the average distance between ratings. These raw agreement statistics must then
 2297 be compared with the rate of chance agreement — the level of agreement that would be
 2298 obtained between two annotators who ignored the data.

2299 Cohen's Kappa is widely used for quantifying the agreement on discrete labeling tasks (Cohen,
 2300 1960; Carletta, 1996),¹²

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

2301 The numerator is the difference between the observed agreement and the chance agreement,
 2302 and the denominator is the difference between perfect agreement and chance agreement.
 2303 Thus, $\kappa = 1$ when the annotators agree in every case, and $\kappa = 0$ when the annotators agree
 2304 only as often as would happen by chance. Various heuristic scales have been proposed
 2305 for determining when κ indicates “moderate”, “good”, or “substantial” agreement; for
 2306 reference, Lee and Narayanan (2005) report $\kappa \approx 0.45 - 0.47$ for annotations of emotions
 2307 in spoken dialogues, which they describe as “moderate agreement”; Stolcke et al. (2000)
 2308 report $\kappa = 0.8$ for annotations of dialogue acts, which are labels for the purpose of each
 2309 turn in a conversation.

2310 When there are two annotators, the expected chance agreement is computed as,

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

2311 where k is a sum over labels, and $\hat{\Pr}(Y = k)$ is the empirical probability of label k across
 2312 all annotations. The formula is derived from the expected number of agreements if the
 2313 annotations were randomly shuffled. Thus, in a binary labeling task, if one label is applied
 2314 to 90% of instances, chance agreement is $.9^2 + .1^2 = .82$.

2315 4.5.2.2 Crowdsourcing

2316 Crowdsourcing is often used to rapidly obtain annotations for classification problems. For
 2317 example, Amazon Mechanical Turk makes it possible to define “human intelligence tasks
 2318 (hits)”, such as labeling data. The researcher sets a price for each set of annotations
 2319 and a list of minimal qualifications for annotators, such as their native language and
 2320 their satisfaction rate on previous tasks. The use of relatively untrained “crowdworkers”

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

contrasts with earlier annotation efforts, which relied on professional linguists (Marcus et al., 1993). However, crowdsourcing has been found to produce reliable annotations for many language-related tasks (Snow et al., 2008). Crowdsourcing is part of the broader field of human computation (Law and Ahn, 2011).

Additional resources

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

Exercises

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

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

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

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

- Prove that F-measure is never greater than the arithmetic mean of recall and precision, $\frac{r+p}{2}$. Your solution should also show that F-measure is equal to $\frac{r+p}{2}$ iff $r = p$.
- Given a binary classification problem in which the probability of the “positive” label is equal to α , what is the expected F-measure of a random classifier which ignores the data, and selects $\hat{y} = +1$ with probability $\frac{1}{2}$? (Assume that $p(\hat{y}) \perp p(y)$.) What is the expected F-measure of a classifier that selects $\hat{y} = +1$ with probability α (also independent of $y^{(i)}$)? Depending on α , which random classifier will score better?
- Suppose that binary classifiers c_1 and c_2 disagree on $N = 30$ cases, and that c_1 is correct in $k = 10$ of those cases.
 - Write a program that uses primitive functions such as `exp` and `factorial` to compute the two-tailed p -value — you may use an implementation of the “choose” function if one is available. Verify your code against the output of a library for computing the binomial test or the binomial CDF, such as `scipy.stats.binom` in Python.

- 2351 • Then use a randomized test to try to obtain the same p -value. In each sample,
 2352 draw from a binomial distribution with $N = 30$ and $\theta = \frac{1}{2}$. Count the fraction of
 2353 samples in which $k \leq 10$. This is the one-tailed p -value; double this to compute
 2354 the two-tailed p -value.
- 2355 • Try this with varying numbers of bootstrap samples: $M \in \{100, 1000, 5000, 10000\}$.
 2356 For $M = 100$ and $M = 1000$, run the test 10 times, and plot the resulting
 2357 p -values.
- 2358 • Finally, perform the same tests for $N = 70$ and $k = 25$.
- 2359 5. SemCor 3.0 is a labeled dataset for word sense disambiguation. You can download
 2360 it,¹³ or access it in `nltk.corpora.semcor`.
 2361 Choose a word that appears at least ten times in SemCor (find), and annotate
 2362 its WordNet senses across ten randomly-selected examples, without looking at the
 2363 ground truth. Use online WordNet to understand the definition of each of the
 2364 senses.¹⁴ Have a partner do the same annotations, and compute the raw rate of
 2365 agreement, expected chance rate of agreement, and Cohen's kappa.
- 2366 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 randomly-selected 400
 2367 reviews as a test set.
 2368
 2369 Download a sentiment lexicon, such as the one currently available from Bing Liu,
 2370 <https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html>. Tokenize the data, and
 2371 classify each document as positive iff it has more positive sentiment words than
 2372 negative sentiment words. Compute the accuracy and F -measure on detecting positive
 2373 reviews on the test set, using this lexicon-based classifier.
 2374
 2375 Then train a discriminative classifier (averaged perceptron or logistic regression) on
 2376 the training set, and compute its accuracy and F -measure on the test set.
 2377
 2378 Determine whether the differences are statistically significant, using two-tailed hypothesis
 2379 tests: Binomial for the difference in accuracy, and bootstrap for the difference in
 2380 macro- F -measure.
- 2381 The remaining problems will require you to build a classifier and test its properties. Pick
 2382 a multi-class text classification dataset, such as RCV1¹⁵). Divide your data into training
 2383 (60%), development (20%), and test sets (20%), if no such division already exists. [todo:
 2384 this dataset is already tokenized, find something else]

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

2383 7. Compare various vocabulary sizes of $10^2, 10^3, 10^4, 10^5$, using the most frequent words
2384 in each case (you may use any reasonable tokenizer). Train logistic regression classifiers
2385 for each vocabulary size, and apply them to the development set. Plot the accuracy
2386 and Macro- F -measure with the increasing vocabulary size. For each vocabulary size,
2387 tune the regularizer to maximize accuracy on a subset of data that is held out from
2388 the training set.

2389 8. Compare the following tokenization algorithms:

- 2390 • Whitespace, using a regular expression
- 2391 • Penn Treebank
- 2392 • Split input into five-character units, regardless of whitespace or punctuation

2393 Compute the token/type ratio for each tokenizer on the training data, and explain
2394 what you find. Train your classifier on each tokenized dataset, tuning the regularizer
2395 on a subset of data that is held out from the training data. Tokenize the development
2396 set, and report accuracy and Macro- F -measure.

2397 9. Apply the Porter and Lancaster stemmers to the training set, using any reasonable
2398 tokenizer, and compute the token/type ratios. Train your classifier on the stemmed
2399 data, and compute the accuracy and Macro- F -measure on stemmed development
2400 data, again using a held-out portion of the training data to tune the regularizer.

2401 10. Identify the best combination of vocabulary filtering, tokenization, and stemming
2402 from the previous three problems. Apply this preprocessing to the test set, and
2403 compute the test set accuracy and Macro- F -measure. Compare against a baseline
2404 system that applies no vocabulary filtering, whitespace tokenization, and no stemming.

2405 Use the binomial test to determine whether your best-performing system is significantly
2406 more accurate than the baseline.

2407 Use the bootstrap test with $M = 10^4$ to determine whether your best-performing
2408 system achieves significantly higher macro- F -measure.

2409 Chapter 5

2410 Learning without supervision

2411 So far we've assumed the following setup:

- 2412 • a training set where you get observations \mathbf{x} and labels y ;
2413 • a test set where you only get observations \mathbf{x} .

2414 Without labeled data, is it possible to learn anything? This scenario is known as unsupervised
2415 learning, and we will see that indeed it is possible to learn about the underlying structure of
2416 unlabeled observations. This chapter will also explore some related scenarios: semi-supervised
2417 learning, in which only some instances are labeled, and domain adaptation, in which the
2418 training data differs from the data on which the trained system will be deployed.

2419 5.1 Unsupervised learning

2420 To motivate unsupervised learning, consider the problem of word sense disambiguation
2421 (\S 4.2). Our goal is to classify each instance of a word, such as bank into a sense,

- 2422 • bank#1: a financial institution
2423 • bank#2: the land bordering a river

2424 It is difficult to obtain sufficient training data for word sense disambiguation, because even
2425 a large corpus will contain only a few instances of all but the most common words. Is it
2426 possible to learn anything about these different senses without labeled data?

2427 Word sense disambiguation is usually performed using feature vectors constructed from
2428 the local context of the word to be disambiguated. For example, for the word bank, the
2429 immediate context might typically include words from one of the following two groups:

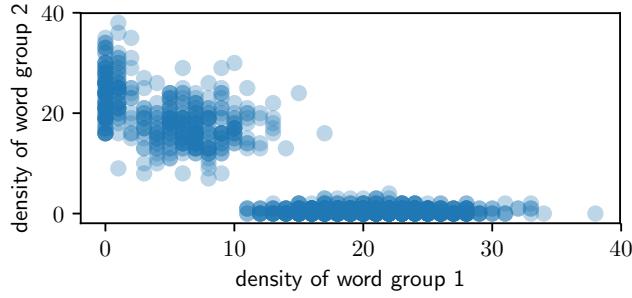


Figure 5.1: Counts of words from two different context groups

- 2430 1. financial, deposits, credit, lending, capital, markets, regulated, reserve, liquid, assets
 2431 2. land, water, geography, stream, river, flow, deposits, discharge, channel, ecology

2432 Now consider a scatterplot, in which each point is a document containing the word bank.
 2433 The location of the document on the x -axis is the count of words in group 1, and the
 2434 location on the y -axis is the count for group 2. In such a plot, shown in Figure 5.1, two
 2435 “blobs” might emerge, and these blobs correspond to the different senses of bank.

2436 Here’s a related scenario, from a different problem. Suppose you download thousands
 2437 of news articles, and make a scatterplot, where each point corresponds to a document: the
 2438 x -axis is the frequency of the group of words (hurricane, winds, storm); the y -axis is the
 2439 frequency of the group (election, voters, vote). This time, three blobs might emerge: one
 2440 for documents that are largely about a hurricane, another for documents largely about a
 2441 election, and a third for documents about neither topic.

2442 These clumps represent the underlying structure of the data. But the two-dimensional
 2443 scatter plots are based on groupings of context words, and in real scenarios these word lists
 2444 are unknown. Unsupervised learning applies the same basic idea, but in a high-dimensional
 2445 space with one dimension for every context word. This space can’t be directly visualized,
 2446 but the idea is the same: try to identify the underlying structure of the observed data,
 2447 such that there are a few clusters of points, each of which is internally coherent. Clustering
 2448 algorithms are capable of finding such structure automatically.

2449 5.1.1 K -means clustering

2450 Clustering algorithms assign each data point to a discrete cluster, $z_i \in 1, 2, \dots, K$. One of
 2451 the best known clustering algorithms is K -means, an iterative algorithm that maintains
 2452 a cluster assignment for each instance, and a central (“mean”) location for each cluster.
 2453 K -means iterates between updates to the assignments and the centers:

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

```

- 2454 1. each instance is placed in the cluster with the closest center;
 2455 2. each center is recomputed as the average over points in the cluster.

2456 This is formalized in Algorithm 8. The term $\|\mathbf{x}^{(i)} - \boldsymbol{\nu}\|^2$ refers to the squared Euclidean
 2457 norm, $\sum_{j=1}^V (x_j^{(i)} - \nu_j)^2$.

2458 Soft K -means is a particularly relevant variant. Instead of directly assigning each point
 2459 to a specific cluster, soft K -means assigns each point a distribution over clusters $\mathbf{q}^{(i)}$, so
 2460 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 the
 2461 distance of $\mathbf{x}^{(i)}$ to the cluster center $\boldsymbol{\nu}_k$. In turn, the center of each cluster is computed
 2462 from a weighted average of the points in the cluster,

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

2463 We will now explore a probabilistic version of soft K -means clustering, based on expectation
 2464 maximization (EM). Because EM clustering can be derived as an approximation to maximum-likelihood
 2465 estimation, it can be extended in a number of useful ways.

2466 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} :

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

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

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

2467 To estimate the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$, we can maximize the marginal likelihood in
 2468 Equation 5.5. Why is this the right thing to maximize? Without labels, discriminative
 2469 learning is impossible — there's nothing to discriminate. So maximum likelihood is all we
 2470 have.

2471 When the labels are observed, we can estimate the parameters of the Naïve Bayes
 2472 probability model separately for each label. But marginalizing over the labels couples
 2473 these parameters, making direct optimization of $\log p(\mathbf{x})$ intractable. We will approximate
 2474 the log-likelihood by introducing an auxiliary variable $\mathbf{q}^{(i)}$, which is a distribution over the
 2475 label set $\mathcal{Z} = \{1, 2, \dots, K\}$. The optimization procedure will alternate between updates to
 2476 \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 $\frac{q^{(i)}(z)}{q^{(i)}(z)} = 1$,

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

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

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

2477 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
 2478 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.

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

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

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

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 $\boldsymbol{\mu}$ and $\boldsymbol{\phi}$.

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 $q^{(i)}$:

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

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

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

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

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

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

2490 After normalizing, each $\mathbf{q}^{(i)}$ — which is the soft distribution over clusters for data $\mathbf{x}^{(i)}$ — is
 2491 set to the posterior probability $p(z | \mathbf{x}^{(i)}; \boldsymbol{\phi}, \boldsymbol{\mu})$ under the current parameters. Although the
 2492 Lagrange multipliers $\lambda^{(i)}$ were introduced as additional parameters, they drop out during
 2493 normalization.

2494 5.1.2.2 The M-step

Next, we hold fixed the soft assignments $\mathbf{q}^{(i)}$, 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,

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

2495 The term $\log p(\mathbf{x}^{(i)} | z; \boldsymbol{\phi})$ is the conditional log-likelihood for the multinomial, which
 2496 expands to,

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

2497 where C is a constant with respect to $\boldsymbol{\phi}$ — see Equation 2.12 in § 2.1 for more discussion
 2498 of this probability function.

Setting the derivative of J_ϕ equal to zero,

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

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

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

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

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

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

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

5.1.3 EM as an optimization algorithm

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

The fact that there is no guarantee of global optimality means that initialization is important: where you start can determine where you finish. To illustrate this point, Figure 5.2 shows the objective function for EM with ten different random initializations: while the objective function improves monotonically in each run, it converges to several

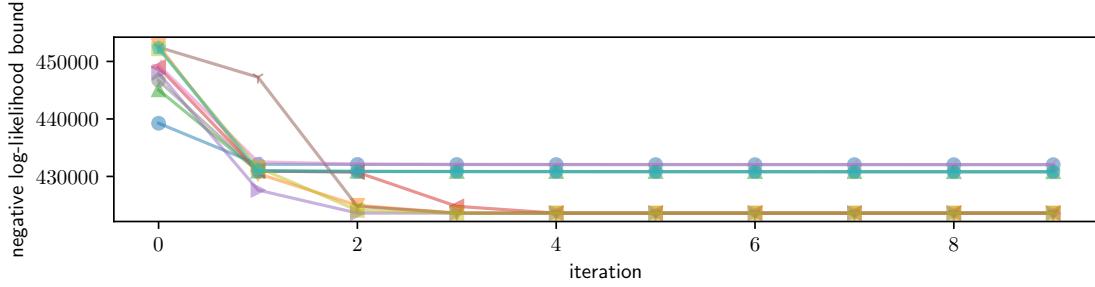


Figure 5.2: Sensitivity of expectation maximization to initialization. Each line shows the progress of optimization from a different random initialization.

2523 different values.¹ For the convex objectives that we encountered in chapter 2, it was not
 2524 necessary to worry about initialization, because gradient-based optimization guaranteed to
 2525 reach the global minimum. But in expectation-maximization — and in the deep neural
 2526 networks from chapter 3 — initialization matters.

2527 In hard EM, each $\mathbf{q}^{(i)}$ distribution assigns probability of 1 to a single label $\hat{z}^{(i)}$, and
 2528 zero probability to all others (Neal and Hinton, 1998). This is similar in spirit to K -means
 2529 clustering, and can outperform standard EM in some cases (Spitkovsky et al., 2010).
 2530 Another variant of expectation maximization incorporates stochastic gradient descent (SGD):
 2531 after performing a local E-step at each instance $\mathbf{x}^{(i)}$, we immediately make a gradient
 2532 update to the parameters $(\boldsymbol{\mu}, \boldsymbol{\phi})$. This algorithm has been called incremental expectation
 2533 maximization (Neal and Hinton, 1998) and online expectation maximization (Sato and
 2534 Ishii, 2000; Cappé and Moulines, 2009), and is especially useful when there is no closed-form
 2535 optimum for the likelihood $p(\mathbf{x} | z)$, and in online settings where new data is constantly
 2536 streamed in (see Liang and Klein, 2009, for a comparison for online EM variants).

2537 5.1.4 How many clusters?

2538 So far, we have assumed that the number of clusters K is given. In some cases, this
 2539 assumption is valid. For example, a lexical semantic resource like WordNet might define
 2540 the number of senses for a word. In other cases, the number of clusters could be a parameter
 2541 for the user to tune: some readers want a coarse-grained clustering of news stories into
 2542 three or four clusters, while others want a fine-grained clustering into twenty or more. But
 2543 many times there is little extrinsic guidance for how to choose K .

2544 One solution is to choose the number of clusters to maximize a metric of clustering
 2545 quality. The other parameters $\boldsymbol{\mu}$ and $\boldsymbol{\phi}$ are chosen to maximize the log-likelihood bound J ,

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

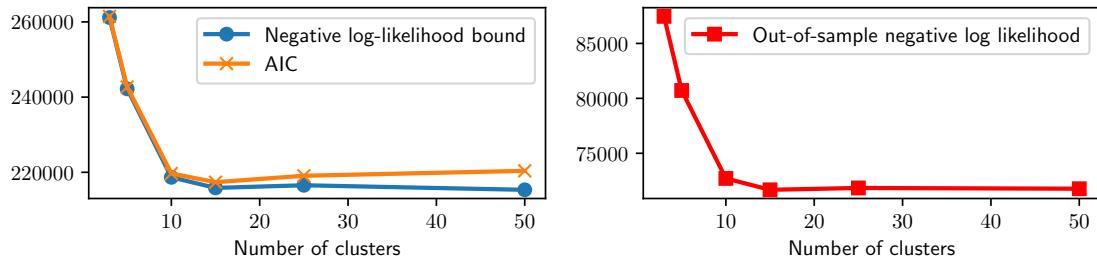


Figure 5.3: The negative log-likelihood and AIC for several runs of expectation maximization, on synthetic data. Although the data was generated from a model with $K = 10$, the optimal number of clusters is $\hat{K} = 15$, according to AIC and the heldout log-likelihood. The training set log-likelihood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

so this might seem a potential candidate for tuning K . However, J will never decrease with K : if it is possible to obtain a bound of J_K with K clusters, then it is always possible to do at least as well with $K+1$ clusters, by simply ignoring the additional cluster and setting its probability to zero in \mathbf{q} and $\boldsymbol{\mu}$. It is therefore necessary to introduce a penalty for model complexity, so that fewer clusters are preferred. For example, the Akaike Information Criterion (AIC; Akaike, 1974) is the linear combination of the number of parameters and the log-likelihood,

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

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

Another choice is to maximize the predictive likelihood on heldout data. This data is not used to estimate the model parameters ϕ and $\boldsymbol{\mu}$, and so it is not the case that the likelihood on this data is guaranteed to increase with K . Figure 5.3 shows the negative log-likelihood on training and heldout data, as well as the AIC.

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

2569 5.2 Applications of expectation-maximization

2570 EM is not really an “algorithm” like, say, quicksort. Rather, it is a framework for learning
 2571 with missing data. The recipe for using EM on a problem of interest is:

- 2572 • Introduce latent variables \mathbf{z} , such that it is easy to write the probability $P(\mathbf{x}, \mathbf{z})$. It
 2573 should also be easy to estimate the associated parameters, given knowledge of \mathbf{z} .
- 2574 • Derive the E-step updates for $q(\mathbf{z})$, which is typically factored as $q(\mathbf{z}) = \prod_{i=1}^N q_{z^{(i)}}(z^{(i)})$,
 2575 where i is an index over instances.
- 2576 • The M-step updates typically correspond to the soft version of a probabilistic supervised
 2577 learning algorithm, like Naïve Bayes.

2578 This section discusses a few of the many applications of this general framework.

2579 5.2.1 Word sense induction

2580 The chapter began by considering the problem of word sense disambiguation when the
 2581 senses are not known in advance. Expectation-maximization can be applied to this problem
 2582 by treating each cluster as a word sense. Each instance represents the use of an ambiguous
 2583 word, and $\mathbf{x}^{(i)}$ is a vector of counts for the other words that appear nearby: Schütze (1998)
 2584 uses all words within a 50-word window. The probability $p(\mathbf{x}^{(i)} | z)$ can be set to the
 2585 multinomial distribution, as in Naïve Bayes. The EM algorithm can be applied directly to
 2586 this data, yielding clusters that (hopefully) correspond to the word senses.

Better performance can be obtained by first applying truncated singular value decomposition
 (SVD) to the matrix of context-counts $\mathbf{C}_{ij} = \text{count}(i, j)$, where $\text{count}(i, j)$ is the count of
 word j in the context of instance i . Truncated singular value decomposition approximates
 the matrix \mathbf{C} as a product of three matrices, $\mathbf{U}, \mathbf{S}, \mathbf{V}$, under the constraint that \mathbf{U} and \mathbf{V}
 are orthonormal, and \mathbf{S} is diagonal:

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

2587 where $\|\cdot\|_F$ is the Frobenius norm, $\|X\|_F = \sqrt{\sum_{i,j} X_{i,j}^2}$. The matrix \mathbf{U} contains the
 2588 left singular vectors of \mathbf{C} , and the rows of this matrix can be used as low-dimensional
 2589 representations of the count vectors \mathbf{c}_i . EM clustering can be made more robust by setting
 2590 the instance descriptions $\mathbf{x}^{(i)}$ equal to these rows, rather than using raw counts (Schütze,

2591 1998). However, because the instances are now dense vectors of continuous numbers, the
 2592 probability $p(\mathbf{x}^{(i)} | z)$ must be defined as a multivariate Gaussian distribution.

2593 In truncated singular value decomposition, the hyperparameter K is the truncation
 2594 limit: when K is equal to the rank of \mathbf{C} , the norm of the difference between the original
 2595 matrix \mathbf{C} and its reconstruction \mathbf{USV}^\top will be zero. Lower values of K increase the
 2596 reconstruction error, but yield vector representations that are smaller and easier to learn
 2597 from. Singular value decomposition is discussed in more detail in chapter 14.

2598 5.2.2 Semi-supervised learning

2599 Expectation-maximization can also be applied to the problem of semi-supervised learning:
 2600 learning from both labeled and unlabeled data in a single model. Semi-supervised learning
 2601 makes use of ground truth annotations, ensuring that each label y corresponds to the
 2602 desired concept. By adding unlabeled data, it is possible cover a greater fraction of the
 2603 features than would be possible using labeled data alone. Other methods for semi-supervised
 2604 learning are discussed in § 5.3, but for now, let's approach the problem within the framework
 2605 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,

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

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

2606 The left sum is identical to the objective in Naïve Bayes; the right sum is the marginal
 2607 log-likelihood for expectation-maximization clustering, from Equation 5.5. We can construct
 2608 a lower bound on this log-likelihood by introducing distributions $\mathbf{q}^{(j)}$ for all $j \in \{N_\ell + 1, \dots, N_\ell + N_u\}$.
 2609 The E-step updates these distributions; the M-step updates the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$, using
 2610 the expected counts from the unlabeled data and the observed counts from the labeled data.

2611 A critical issue in semi-supervised learning is how to balance the impact of the labeled
 2612 and unlabeled data on the classifier weights, especially when the unlabeled data is much
 2613 larger than the labeled dataset. The risk is that the unlabeled data will dominate, causing
 2614 the parameters to drift towards a “natural clustering” of the instances — which may not
 2615 correspond to a good classifier for the labeled data. One solution is to heuristically reweight
 2616 the two components of Equation 5.26, tuning the weight of the two components on a heldout
 2617 development set (Nigam et al., 2000).

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

```

for Document  $i \in \{1, 2, \dots, N\}$  do:
  Draw the label  $y^{(i)} \sim \text{Categorical}(\boldsymbol{\mu})$ ;
  Draw the component  $z^{(i)} \sim \text{Categorical}(\boldsymbol{\beta}_{y^{(i)}})$ ;
  Draw the word counts  $\mathbf{x}^{(i)} | y^{(i)}, z^{(i)} \sim \text{Multinomial}(\boldsymbol{\phi}_{z^{(i)}})$ .
  
```

2618 5.2.3 Multi-component modeling

2619 As a final application, let's return to fully supervised classification. A classic dataset
 2620 for text classification is 20 newsgroups, which contains posts to a set of online forums,
 2621 called newsgroups. One of the newsgroups is comp.sys.mac.hardware, which discusses
 2622 Apple computing hardware. Suppose that within this newsgroup there are two kinds of
 2623 posts: reviews of new hardware, and question-answer posts about hardware problems. The
 2624 language in these components of the mac.hardware class might have little in common; if so,
 2625 it would be better to model these components separately, rather than treating their union
 2626 as a single class. However, the component responsible for each instance is not directly
 2627 observed.

2628 Recall that Naïve Bayes is based on a generative process, which provides a stochastic
 2629 explanation for the observed data. In Naïve Bayes, each label is drawn from a categorical
 2630 distribution with parameter $\boldsymbol{\mu}$, and each vector of word counts is drawn from a multinomial
 2631 distribution with parameter $\boldsymbol{\phi}_y$. For multi-component modeling, we envision a slightly
 2632 different generative process, incorporating both the observed label $y^{(i)}$ and the latent
 2633 component $z^{(i)}$. This generative process is shown in Algorithm 9. A new parameter $\boldsymbol{\beta}_{y^{(i)}}$
 2634 defines the distribution of components, conditioned on the label $y^{(i)}$. The component, and
 2635 not the class label, then parametrizes the distribution over words.

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 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

-
- (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 tiraillé entre deux personnages et directions qui ne parviennent pas à coexister en harmonie.³
- (5.3) Denis Villeneuve a réussi une suite parfaitement 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 parfaitement 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 distribution over the missing data, $q_{Z|Y}^{(i)}$. The M-step updates the parameters,

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

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

2636 5.3 Semi-supervised learning

2637 In semi-supervised learning, the learner makes use of both labeled and unlabeled data. To
 2638 see how this could help, suppose you want to do sentiment analysis in French. In Table 5.1,
 2639 there are two labeled examples, one positive and one negative. From this data, a learner
 2640 could conclude that réussi is positive and long is negative. This isn't much! However, we
 2641 can propagate this information to the unlabeled data, and potentially learn more.

- 2642 • If we are confident that réussi is positive, then we might guess that (5.3) is also
 2643 positive.
- 2644 • That suggests that parfaitement is also positive.
- 2645 • We can then propagate this information to (5.5), and learn from this words in this
 2646 example.

- 2647 • Similarly, we can propagate from the labeled data to (5.4), which we guess to be
 2648 negative because it shares the word long. This suggests that bavard is also negative,
 2649 which we propagate to (5.6).

2650 Instances (5.3) and (5.4) were “similar” to the labeled examples for positivity and negativity,
 2651 respectively. By using these instances to expand the models for each class, it became
 2652 possible to correctly label instances (5.5) and (5.6), which didn’t share any important
 2653 features with the original labeled data. This requires a key assumption: that similar
 2654 instances will have similar labels.

2655 In § 5.2.2, we discussed how expectation maximization can be applied to semi-supervised
 2656 learning. Using the labeled data, the initial parameters ϕ would assign a high weight for
 2657 réussи in the positive class, and a high weight for long in the negative class. These weights
 2658 helped to shape the distributions q for instances (5.3) and (5.4) in the E-step. In the next
 2659 iteration of the M-step, the parameters ϕ are updated with counts from these instances,
 2660 making it possible to correctly label the instances (5.5) and (5.6).

2661 However, expectation-maximization has an important disadvantage: it requires using a
 2662 generative classification model, which restricts the features that can be used for classification.
 2663 In this section, we explore non-probabilistic approaches, which impose fewer restrictions
 2664 on the classification model.

2665 5.3.1 Multi-view learning

2666 EM semi-supervised learning can be viewed as self-training: the labeled data guides the
 2667 initial estimates of the classification parameters; these parameters are used to compute
 2668 a label distribution over the unlabeled instances, $q^{(i)}$; the label distributions are used
 2669 to update the parameters. The risk is that self-training drifts away from the original
 2670 labeled data. This problem can be ameliorated by multi-view learning. Here we take the
 2671 assumption that the features can be decomposed into multiple “views”, each of which is
 2672 conditionally independent, given the label. For example, consider the problem of classifying
 2673 a name as a person or location: one view is the name itself; another is the context in which
 2674 it appears. This situation is illustrated in Table 5.2.

2675 Co-training is an iterative multi-view learning algorithm, in which there are separate
 2676 classifiers for each view (Blum and Mitchell, 1998). At each iteration of the algorithm,
 2677 each classifier predicts labels for a subset of the unlabeled instances, using only the features
 2678 available in its view. These predictions are then used as ground truth to train the classifiers
 2679 associated with the other views. In the example shown in Table 5.2, the classifier on $x^{(1)}$
 2680 might correctly label instance #5 as a person, because of the feature Dr; this instance
 2681 would then serve as training data for the classifier on $x^{(2)}$, which would then be able to
 2682 correctly label instance #6, thanks to the feature recommended. If the views are truly

	$\mathbf{x}^{(1)}$	$\mathbf{x}^{(2)}$	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

2683 independent, this procedure is robust to drift. Furthermore, it imposes no restrictions on
 2684 the classifiers that can be used for each view.

2685 Word-sense disambiguation is particularly suited to multi-view learning, thanks to the
 2686 heuristic of “one sense per discourse”: if a polysemous word is used more than once in
 2687 a given text or conversation, all usages refer to the same sense (Gale et al., 1992). This
 2688 motivates a multi-view learning approach, in which one view corresponds to the local
 2689 context (the surrounding words), and another view corresponds to the global context at
 2690 the document level (Yarowsky, 1995). The local context view is first trained on a small
 2691 seed dataset. We then identify its most confident predictions on unlabeled instances. The
 2692 global context view is then used to extend these confident predictions to other instances
 2693 within the same documents. These new instances are added to the training data to the
 2694 local context classifier, which is retrained and then applied to the remaining unlabeled
 2695 data.

2696 5.3.2 Graph-based algorithms

2697 Another family of approaches to semi-supervised learning begins by constructing a graph,
 2698 in which pairs of instances are linked with symmetric weights $\omega_{i,j}$, e.g.,

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

2699 The goal is to use this weighted graph to propagate labels from a small set of labeled
 2700 instances to larger set of unlabeled instances.

2701 In label propagation, this is done through a series of matrix operations (Zhu et al.,
 2702 2003). Let \mathbf{Q} be a matrix of size $N \times K$, in which each row $\mathbf{q}^{(i)}$ describes the labeling
 2703 of instance i . When ground truth labels are available, then $\mathbf{q}^{(i)}$ is an indicator vector,
 2704 with $q_{y^{(i)}}^{(i)} = 1$ and $q_{y' \neq y^{(i)}}^{(i)} = 0$. Let us refer to the submatrix of rows containing labeled
 2705 instances as \mathbf{Q}_L , and the remaining rows as \mathbf{Q}_U . The rows of \mathbf{Q}_U are initialized to assign
 2706 equal probabilities to all labels, $q_{i,k} = \frac{1}{K}$.

2707 Now, let $T_{i,j}$ represent the “transition” probability of moving from node j to node i ,

$$T_{i,j} \triangleq \Pr(j \rightarrow i) = \frac{\omega_{i,j}}{\sum_{k=1}^N \omega_{k,j}}. \quad [5.33]$$

We compute values of $T_{i,j}$ for all instances j and all unlabeled instances i , forming a matrix of size $N_U \times N$. If the dataset is large, this matrix may be expensive to store and manipulate; a solution is to sparsify it, by keeping only the κ largest values in each row, and setting all other values to zero. We can then “propagate” the label distributions to the unlabeled instances,

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

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

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

2708 The expression $\tilde{\mathbf{Q}}_U \mathbf{1}$ indicates multiplication of $\tilde{\mathbf{Q}}_U$ by a column vector of ones, which is
 2709 equivalent to computing the sum of each row of $\tilde{\mathbf{Q}}_U$. The matrix $\text{Diag}(\mathbf{s})$ is a diagonal
 2710 matrix with the elements of \mathbf{s} on the diagonals. The product $\text{Diag}(\mathbf{s})^{-1} \tilde{\mathbf{Q}}_U$ has the effect
 2711 of normalizing the rows of $\tilde{\mathbf{Q}}_U$, so that each row of \mathbf{Q}_U is a probability distribution over
 2712 labels.

2713 5.4 Domain adaptation

2714 In many practical scenarios, the labeled data differs in some key respect from the data
 2715 to which the trained model is to be applied. A classic example is in consumer reviews:
 2716 we may have labeled reviews of movies (the source domain), but we want to predict the
 2717 reviews of appliances (the target domain). A similar issues arise with genre differences:
 2718 most linguistically-annotated data is news text, but application domains range from social
 2719 media to electronic health records. In general, there may be several source and target
 2720 domains, each with their own properties; however, for simplicity, this discussion will focus
 2721 mainly on the case of a single source and target domain.

2722 The simplest approach is “direct transfer”: train a classifier on the source domain, and
 2723 apply it directly to the target domain. The accuracy of this approach depends on the
 2724 extent to which features are shared across domains. In review text, words like outstanding
 2725 and disappointing will apply across both movies and appliances; but others, like terrifying,
 2726 may have meanings that are domain-specific. Domain adaptation algorithms attempt to
 2727 do better than direct transfer, by learning from data in both domains. There are two
 2728 main families of domain adaptation algorithms, depending on whether any labeled data is
 2729 available in the target domain.

2730 5.4.1 Supervised domain adaptation

2731 In supervised domain adaptation, there is a small amount of labeled data in the target
 2732 domain, and a large amount of data in the source domain. The simplest approach would
 2733 be to ignore domain differences, and simply merge the training data from the source
 2734 and target domains. There are several other baseline approaches to dealing with this
 2735 scenario (Daumé III, 2007):

2736 Interpolation. Train a classifier for each domain, and combine their predictions. For
 2737 example,

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

2738 where Ψ_s and Ψ_t are the scoring functions from the source and target domain
 2739 classifiers respectively, and λ_s is the interpolation weight.

2740 Prediction. Train a classifier on the source domain data, use its prediction as an additional
 2741 feature in a classifier trained on the target domain data.

2742 Priors. Train a classifier on the source domain data, and use its weights as a prior distribution
 2743 on the weights of the classifier for the target domain data. This is equivalent to
 2744 regularizing the target domain weights towards the weights of the source domain
 2745 classifier (Chelba and Acero, 2006),

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

2746 where $\ell^{(i)}$ is the prediction loss on instance i , and λ is the regularization weight.

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

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

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

2747 with (boring, −, movie) indicating the word boring appearing in a negative labeled document
 2748 in the movie domain, and (boring, −, ∗) indicating the same word in a negative labeled
 2749 document in any domain. It is up to the learner to allocate weight between the domain-specific
 2750 and cross-domain features: for words that facilitate prediction in both domains, the learner
 2751 will use the cross-domain features; for words that are relevant only to a single domain, the
 2752 domain-specific features will be used. Any discriminative classifier can be used with these
 2753 augmented features.⁹

2754 5.4.2 Unsupervised domain adaptation

2755 In unsupervised domain adaptation, there is no labeled data in the target domain. Unsupervised
 2756 domain adaptation algorithms cope with this problem by trying to make the data from
 2757 the source and target domains as similar as possible. This is typically done by learning
 2758 a projection function, which puts the source and target data in a shared space, in which
 2759 a learner can generalize across domains. This projection is learned from data in both
 2760 domains, and is applied to the base features — for example, the bag-of-words in text
 2761 classification. The projected features can then be used both for training and for prediction.

2762 5.4.2.1 Linear projection

2763 In linear projection, the cross-domain representation is constructed by a matrix-vector
 2764 product,

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

2765 The projected vectors $\mathbf{g}(\mathbf{x}^{(i)})$ can then be used as base features during both training (from
 2766 the source domain) and prediction (on the target domain).

2767 The projection matrix \mathbf{U} can be learned in a number of different ways, but many
 2768 approaches focus on compressing and reconstructing the base features (Ando and Zhang,
 2769 2005). For example, we can define a set of pivot features, which are typically chosen because
 2770 they appear in both domains: in the case of review documents, pivot features might include
 2771 evaluative adjectives like outstanding and disappointing (Blitzer et al., 2007). For each
 2772 pivot feature j , we define an auxiliary problem of predicting whether the feature is present
 2773 in each example, using the remaining base features. Let ϕ_j denote the weights of this
 2774 classifier, and us horizontally concatenate the weights for each of the N_p pivot features into
 2775 a matrix $\Phi = [\phi_1, \phi_2, \dots, \phi_{N_p}]$.

2776 We then perform truncated singular value decomposition on Φ , as described in § 5.2.1,
 2777 obtaining $\Phi \approx \mathbf{U}\mathbf{S}\mathbf{V}^\top$. The rows of the matrix \mathbf{U} summarize information about each base
 2778 feature: indeed, the truncated singular value decomposition identifies a low-dimension basis
 2779 for the weight matrix Φ , which in turn links base features to pivot features. Suppose that a

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).

base feature reliable occurs only in the target domain of appliance reviews. Nonetheless, it will have a positive weight towards some pivot features (e.g., outstanding, recommended), and a negative weight towards others (e.g., worthless, unpleasant). A base feature such as watchable might have the same associations with the pivot features, and therefore, $\mathbf{u}_{\text{reliable}} \approx \mathbf{u}_{\text{watchable}}$. The matrix \mathbf{U} can thus project the base features into a space in which this information is shared.

5.4.2.2 Non-linear projection

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

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

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

To formalize this idea, let $d^{(i)}$ represent the domain of instance i , and let $\ell_d(\mathbf{g}(\mathbf{x}^{(i)}), d^{(i)}; \boldsymbol{\theta}_d)$ represent the loss of a classifier (typically a deep neural network) trained to predict $d^{(i)}$ from the transformed representation $\mathbf{g}(\mathbf{x}^{(i)})$, using parameters $\boldsymbol{\theta}_d$. Analogously, let $\ell_y(\mathbf{g}(\mathbf{x}^{(i)}), y^{(i)}; \boldsymbol{\theta}_y)$ represent the loss of a classifier trained to predict the label $y^{(i)}$ from $\mathbf{g}(\mathbf{x}^{(i)})$, using parameters $\boldsymbol{\theta}_y$. The transformation \mathbf{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,

$$\min_{\mathbf{f}, \boldsymbol{\theta}_g, \boldsymbol{\theta}_y, \boldsymbol{\theta}_d} \sum_{i=1}^{N_\ell + N_u} \ell_d(\mathbf{g}(\mathbf{x}^{(i)}; \boldsymbol{\theta}_g), d^{(i)}; \boldsymbol{\theta}_d) - \sum_{i=1}^{N_\ell} \ell_y(\mathbf{g}(\mathbf{x}^{(i)}; \boldsymbol{\theta}_g), y^{(i)}; \boldsymbol{\theta}_y), \quad [5.40]$$

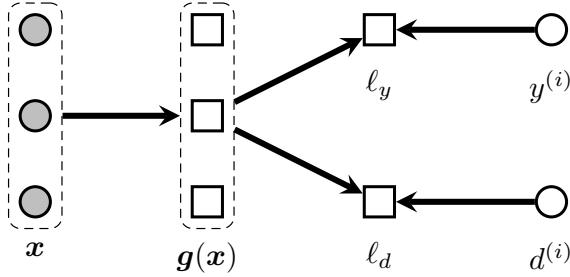


Figure 5.4: A schematic view of adversarial domain adaptation. The loss ℓ_y is computed only for instances from the source domain, where labels $y^{(i)}$ are available.

2812 where N_ℓ is the number of labeled instances and N_u is the number of unlabeled instances,
 2813 with the labeled instances appearing first in the dataset. This setup is shown in Figure 5.4.
 2814 The loss can be optimized by stochastic gradient descent, jointly training the parameters
 2815 of the non-linear transformation θ_g , and the parameters of the prediction models θ_d and
 2816 θ_y .

2817 5.5 *Other approaches to learning with latent variables

2818 Expectation maximization provides a general approach to learning with latent variables,
 2819 but it has limitations. One is the sensitivity to initialization; in practical applications,
 2820 considerable attention may need to be devoted to finding a good initialization. A second
 2821 issue is that EM tends to be easiest to apply in cases where the latent variables have a clear
 2822 decomposition (in the cases we have considered, they decompose across the instances). For
 2823 these reasons, it is worth briefly considering some alternatives to EM.

2824 5.5.1 Sampling

2825 In EM clustering, there is a distribution $q^{(i)}$ for the missing data related to each instance.
 2826 The M-step consists of updating the parameters of this distribution. An alternative is to to
 2827 draw samples of the latent variables. If the sampling distribution is designed correctly, this
 2828 procedure will eventually converge to drawing samples from the true posterior over the
 2829 missing data, $p(z^{(1:N_z)} | x^{(1:N_x)})$. For example, in the case of clustering, the missing data
 2830 $z^{(1:N_z)}$ is the set of cluster memberships, $y^{(1:N)}$, so we draw samples from the posterior
 2831 distribution over clusterings of the data. If a single clustering is required, we can select the
 2832 one with the highest conditional likelihood, $\hat{z} = \text{argmax}_z p(z^{(1:N_z)} | x^{(1:N_x)})$.

This general family of algorithms is called Markov Chain Monte Carlo (MCMC): “Monte Carlo” because it is based on a series of random draws; “Markov Chain” because the sampling procedure must be designed such that each sample depends only on the

previous sample, and not on the entire sampling history. Gibbs sampling is an MCMC algorithm in which each latent variable is sampled from its posterior distribution,

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

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

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

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

2833 In this case, the sampling distribution does not depend on the other instances $\mathbf{x}^{(-i)}, \mathbf{z}^{(-i)}$:
 2834 given the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$, the posterior distribution over each $z^{(i)}$ can be computed
 2835 from $\mathbf{x}^{(i)}$ alone.

2836 In sampling algorithms, there are several choices for how to deal with the parameters.
 2837 One possibility is to sample them too. To do this, we must add them to the generative
 2838 story, by introducing a prior distribution. For the multinomial and categorical parameters
 2839 in the EM clustering model, the Dirichlet distribution is a typical choice, since it defines a
 2840 probability on exactly the set of vectors that can be parameters: vectors that sum to one
 2841 and include only non-negative numbers.¹⁰

2842 To incorporate this prior, the generative model must augmented to indicate that each
 2843 $\boldsymbol{\phi}_z \sim \text{Dirichlet}(\boldsymbol{\alpha}_\phi)$, and $\boldsymbol{\mu} \sim \text{Dirichlet}(\boldsymbol{\alpha}_\mu)$. The hyperparameters $\boldsymbol{\alpha}$ are typically set to
 2844 a constant vector $\boldsymbol{\alpha} = [\alpha, \alpha, \dots, \alpha]$. When α is large, the Dirichlet distribution tends to
 2845 generate vectors that are nearly uniform; when α is small, it tends to generate vectors that
 2846 assign most of their probability mass to a few entries. Given prior distributions over $\boldsymbol{\phi}$ and
 2847 $\boldsymbol{\mu}$, we can now include them in Gibbs sampling, drawing values for these parameters from
 2848 posterior distributions that are conditioned on the other variables in the model.

2849 Unfortunately, sampling $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$ usually leads to slow convergence, meaning that a
 2850 large number of samples is required before the Markov chain breaks free from the initial

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.

2851 conditions. The reason is that the sampling distributions for these parameters are tightly
 2852 constrained by the cluster memberships $\mathbf{y}^{(i)}$, which in turn are tightly constrained by the
 2853 parameters. There are two solutions that are frequently employed:

- 2854 • Empirical Bayesian methods maintain ϕ and μ as parameters rather than latent
 2855 variables. They still employ sampling in the E-step of the EM algorithm, but they
 2856 update the parameters using expected counts that are computed from the samples
 2857 rather than from parametric distributions. This EM-MCMC hybrid is also known
 2858 as Monte Carlo Expectation Maximization (MCEM; Wei and Tanner, 1990), and is
 2859 well-suited for cases in which it is difficult to compute $\mathbf{q}^{(i)}$ directly.
- 2860 • In collapsed Gibbs sampling, we analytically integrate ϕ and μ out of the model.
 2861 The cluster memberships $y^{(i)}$ are the only remaining latent variable; we sample them
 2862 from the compound distribution,

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

2863 For multinomial and Dirichlet distributions, this integral can be computed in closed
 2864 form.

2865 MCMC algorithms are guaranteed to converge to the true posterior distribution over
 2866 the latent variables, but there is no way to know how long this will take. In practice, the
 2867 rate of convergence depends on initialization, just as expectation-maximization depends
 2868 on initialization to avoid local optima. Thus, while Gibbs Sampling and other MCMC
 2869 algorithms provide a powerful and flexible array of techniques for statistical inference in
 2870 latent variable models, they are not a panacea for the problems experienced by EM.

2871 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 $p(\mathbf{x} | \boldsymbol{\phi}, \boldsymbol{\mu})$, as

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

$$= \sum_k^K N\mu_k \boldsymbol{\phi}_k \boldsymbol{\phi}_k^\top \quad [5.49]$$

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

where Φ is formed by horizontally concatenating $\boldsymbol{\phi}_1 \dots \boldsymbol{\phi}_K$, and $\text{Diag}(N\boldsymbol{\mu})$ indicates a diagonal matrix with values $N\mu_k$ at position (k, k) . Setting \mathbf{C} equal to its expectation gives,

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

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

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

After obtaining the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$, the distribution over clusters can be computed from Bayes' rule:

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

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

Additional resources

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

- Active learning: the learner selects unlabeled instances and requests annotations (Settles, 2012).

- Multiple instance learning: labels are applied to bags of instances, with a positive label applied if at least one instance in the bag meets the criterion (Dietterich et al., 1997; Maron and Lozano-Pérez, 1998).
- Constraint-driven learning: supervision is provided in the form of explicit constraints on the learner (Chang et al., 2007; Ganchev et al., 2010).
- Distant supervision: noisy labels are generated from an external resource (Mintz et al., 2009, also see § 17.2.3).
- Multitask learning: the learner induces a representation that can be used to solve multiple classification tasks (Collobert et al., 2011).
- Transfer learning: the learner must solve a classification task that differs from the labeled data (Pan and Yang, 2010).

Expectation maximization was introduced by Dempster et al. (1977), and is discussed in more detail by Murphy (2012). Like most machine learning treatments, Murphy focus on continuous observations and Gaussian likelihoods, rather than the discrete observations typically encountered in natural language processing. Murphy (2012) also includes an excellent chapter on MCMC; for a textbook-length treatment, see Robert and Casella (2013). For still more on Bayesian latent variable models, see Barber (2012), and for applications of Bayesian models to natural language processing, see Cohen (2016). Surveys are available for semi-supervised learning (Zhu and Goldberg, 2009) and domain adaptation (Søgaard, 2013), although both pre-date the current wave of interest in deep learning.

Exercises

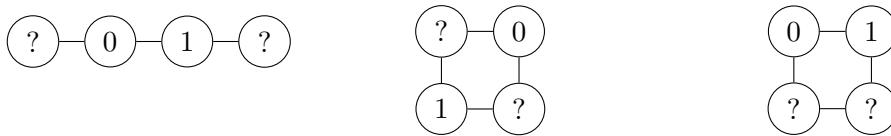
- Derive the expectation maximization update for the parameter μ in the EM clustering model.
- The expectation maximization lower bound \mathcal{J} is defined in Equation 5.10. Prove that the inverse $-\mathcal{J}$ is convex in \mathbf{q} . You can use the following facts about convexity:
 - $f(\mathbf{x})$ 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 $\alpha \in [0, 1]$.
 - 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} .
 - $\log(x + y) \leq \log x + \log y$.
- Derive the E-step and M-step updates for the following generative model. You may assume that the labels $y^{(i)}$ are observed, but $z_m^{(i)}$ is not.
 - For each instance i ,

- 2925 – Draw label $y^{(i)} \sim \text{Categorical}(\boldsymbol{\mu})$
 2926 – For each token $m \in \{1, 2, \dots, M^{(i)}\}$
 2927 * Draw $z_m^{(i)} \sim \text{Categorical}(\boldsymbol{\pi})$
 2928 * If $z_m^{(i)} = 0$, draw the current token from a label-specific distribution,
 2929 $w_m^{(i)} \sim \boldsymbol{\phi}_{y^{(i)}}$
 2930 * If $z_m^{(i)} = 1$, draw the current token from a document-specific distribution,
 2931 $w_m^{(i)} \sim \boldsymbol{\nu}^{(i)}$

2932 4. Use expectation-maximization clustering to train a word-sense induction system,
 2933 applied to the word say.

- 2934 • Import nltk, run nltk.download() and select semcor. Import semcor from nltk.corpus.
- 2935 • The command semcor.tagged_sentences(tag='sense') returns an iterator over
 2936 sense-tagged sentences in the corpus. Each sentence can be viewed as an iterator
 2937 over tree objects. For tree objects that are sense-annotated words, you can
 2938 access the annotation as tree.label(), and the word itself with tree.leaves().
 2939 So semcor.tagged_sentences(tag='sense')[0][2].label() would return the sense
 2940 annotation of the third word in the first sentence.
- 2941 • Extract all sentences containing the senses say.v.01 and say.v.02.
- 2942 • Build bag-of-words vectors $\mathbf{x}^{(i)}$, containing the counts of other words in those
 2943 sentences, including all words that occur in at least two sentences.
- 2944 • Implement and run expectation-maximization clustering on the merged data.
- 2945 • Compute the frequency with which each cluster includes instances of say.v.01
 2946 and say.v.02.

2947 5. Using the iterative updates in Equations 5.34-5.36, compute the outcome of the label
 2948 propagation algorithm for the following examples.



2949 The value inside the node indicates the label, $y^{(i)} \in \{0, 1\}$, with $y^{(i)} = ?$ for unlabeled
 2950 nodes. The presence of an edge between two nodes indicates $w_{i,j} = 1$, and the absence
 2951 of an edge indicates $w_{i,j} = 0$. For the third example, you need only compute the first
 2952 three iterations, and then you can guess at the solution in the limit.

2953 In the remaining exercises, you will try out some approaches for semisupervised learning
 2954 and domain adaptation. You will need datasets in multiple domains. You can obtain

2955 product reviews in multiple domains here: https://www.cs.jhu.edu/~mdredze/datasets/sentiment/processed_acl.tar.gz. Choose a source and target domain, e.g. dvds and books,
 2956 and divide the data for the target domain into training and test sets of equal size.
 2957

- 2958 6. First, quantify the cost of cross-domain transfer.
- 2959 • Train a logistic regression classifier on the source domain training set, and
 2960 evaluate it on the target domain test set.
 2961 • Train a logistic regression classifier on the target domain training set, and
 2962 evaluate it on the target domain test set. This is the “direct transfer” baseline.

2963 Compute the difference in accuracy, which is a measure of the transfer loss across
 2964 domains.

- 2965 7. Next, apply the label propagation algorithm from § 5.3.2.
- 2966 As a baseline, using only 5% of the target domain training set, train a classifier, and
 2967 compute its accuracy on the target domain test set.

2968 Next, apply label propagation:

- 2969 • Compute the label matrix \mathbf{Q}_L for the labeled data (5% of the target domain
 2970 training set), with each row equal to an indicator vector for the label (positive
 2971 or negative).
- 2972 • Iterate through the target domain instances, including both test and training
 2973 data. At each instance i , compute all w_{ij} , using Equation 5.32, with $\alpha = 0.01$.
 2974 Use these values to fill in column i of the transition matrix \mathbf{T} , setting all but the
 2975 ten largest values to zero for each column i . Be sure to normalize the column
 2976 so that the remaining values sum to one. You may need to use a sparse matrix
 2977 for this to fit into memory.
- 2978 • Apply the iterative updates from Equations 5.34-5.36 to compute the outcome
 2979 of the label propagation algorithm for the unlabeled examples.

2980 Select the test set instances from \mathbf{Q}_U , and compute the accuracy of this method.
 2981 Compare with the supervised classifier trained only on the 5% sample of the target
 2982 domain training set.

- 2983 8. Using only 5% of the target domain training data (and all of the source domain
 2984 training data), implement one of the supervised domain adaptation baselines in
 2985 § 5.4.1. See if this improves on the “direct transfer” baseline from the previous
 2986 problem
- 2987 9. Implement EasyAdapt (§ 5.4.1), again using 5% of the target domain training data
 2988 and all of the source domain data.

2989 10. Now try unsupervised domain adaptation, using the “linear projection” method
2990 described in § 5.4.2. Specifically:

- 2991 • Identify 500 pivot features as the words with the highest frequency in the
2992 (complete) training data for the source and target domains. Specifically, let
2993 x_i^d be the count of the word i in domain d : choose the 500 words with the
2994 largest values of $\min(x_i^{\text{source}}, x_i^{\text{target}})$.
- 2995 • Train a classifier to predict each pivot feature from the remaining words in the
2996 document.
- 2997 • Arrange the features of these classifiers into a matrix Φ , and perform truncated
2998 singular value decomposition, with $k = 20$
- 2999 • Train a classifier from the source domain data, using the combined features $\mathbf{x}^{(i)} \oplus$
3000 $\mathbf{U}^\top \mathbf{x}^{(i)}$ — these include the original bag-of-words features, plus the projected
3001 features.
- 3002 • Apply this classifier to the target domain test set, and compute the accuracy.

3003

Part II

3004

Sequences and trees

3005 Chapter 6

3006 Language models

3007 In probabilistic classification, the problem is to compute the probability of a label, conditioned
3008 on the text. Let's now consider the inverse problem: computing the probability of text
3009 itself. Specifically, we will consider models that assign probability to a sequence of word
3010 tokens, $p(w_1, w_2, \dots, w_M)$, with $w_m \in \mathcal{V}$. The set \mathcal{V} is a discrete vocabulary,

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

3011 Why would you want to compute the probability of a word sequence? In many
3012 applications, the goal is to produce word sequences as output:

- 3013 • In machine translation (chapter 18), we convert from text in a source language to
3014 text in a target language.
- 3015 • In speech recognition, we convert from audio signal to text.
- 3016 • In summarization (§ 16.3.4.1; § 19.2), we convert from long texts into short texts.
- 3017 • In dialogue systems (§ 19.3), we convert from the user's input (and perhaps an
3018 external knowledge base) into a text response.

3019 In many of the systems for performing these tasks, there is a subcomponent that
3020 computes the probability of the output text. The purpose of this component is to generate
3021 texts that are more fluent. For example, suppose we want to translate a sentence from
3022 Spanish to English.

3023 (6.1) El cafe negro me gusta mucho.

3024 Here is a literal word-for-word translation (a gloss):

3025 (6.2) The coffee black me pleases much.

3026 A good language model of English will tell us that the probability of this translation is
 3027 low, in comparison with more grammatical alternatives,

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

3028 How can we use this fact? Warren Weaver, one of the early leaders in machine
 3029 translation, viewed it as a problem of breaking a secret code (Weaver, 1955):

3030 When I look at an article in Russian, I say: 'This is really written in English,
 3031 but it has been coded in some strange symbols. I will now proceed to decode.'

3032 This observation motivates a generative model (like Naïve Bayes):

- 3033 • The English sentence $\mathbf{w}^{(e)}$ is generated from a language model, $p_e(\mathbf{w}^{(e)})$.
- 3034 • The Spanish sentence $\mathbf{w}^{(s)}$ is then generated from a translation model, $p_{s|e}(\mathbf{w}^{(s)} | \mathbf{w}^{(e)})$.

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

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

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

3035 This is sometimes called the noisy channel model, because it envisions English text
 3036 turning into Spanish by passing through a noisy channel, $p_{s|e}$. What is the advantage of
 3037 modeling translation this way, as opposed to modeling $p_{e|s}$ directly? The crucial point
 3038 is that the two distributions $p_{s|e}$ (the translation model) and p_e (the language model)
 3039 can be estimated from separate data. The translation model requires examples of correct
 3040 translations, but the language model requires only text in English. Such monolingual data
 3041 is much more widely available. Furthermore, once estimated, the language model p_e can
 3042 be reused in any application that involves generating English text, from summarization to
 3043 speech recognition.

3044 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,

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

3045 This estimator is unbiased: in the theoretical limit of infinite data, the estimate will
 3046 be correct. But in practice, we are asking for accurate counts over an infinite number
 3047 of events, since sequences of words can be arbitrarily long. Even with an aggressive
 3048 upper bound of, say, $M = 20$ tokens in the sequence, the number of possible sequences
 3049 is V^{20} . A small vocabulary for English would have $V = 10^4$, so there are 10^{80} possible
 3050 sequences. Clearly, this estimator is very data-hungry, and suffers from high variance: even
 3051 grammatical sentences will have probability zero if have not occurred in the training data.¹
 3052 We therefore need to introduce bias to have a chance of making reliable estimates from
 3053 finite training data. The language models that follow in this chapter introduce bias in
 3054 various ways.

We begin with n -gram language models, which compute the probability of a sequence as the product of probabilities of subsequences. The probability of a sequence $p(\mathbf{w}) = p(w_1, w_2, \dots, w_M)$ can be refactored using the chain rule (see § A.2):

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

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

Each element in the product is the probability of a word given all its predecessors. We can think of this as a word prediction task: given the context Computers are, we want to compute a probability over the next token. The relative frequency estimate of the probability of the word useless in this context is,

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

3055 We haven't made any approximations yet, and we could have just as well applied the
 3056 chain rule in reverse order,

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

3057 or in any other order. But this means that we also haven't really made any progress: to
 3058 compute the conditional probability $p(w_M | w_{M-1}, w_{M-2}, \dots, w_1)$, we would need to model
 3059 V^{M-1} contexts. Such a distribution cannot be estimated from any realistic sample of text.

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

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

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.

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

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

To compute the probability of an entire sentence, it is convenient to pad the beginning and end with special symbols \square and \blacksquare . Then the bigram ($n = 2$) approximation to the probability of I like black coffee is:

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

3060 This model requires estimating and storing the probability of only V^n events, which is
 3061 exponential in the order of the n -gram, and not V^M , which is exponential in the length of
 3062 the sentence. The n -gram probabilities can be computed by relative frequency estimation,

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

3063 The hyperparameter n controls the size of the context used in each conditional probability.
 3064 If this is misspecified, the language model will perform poorly. Let's consider the potential
 3065 problems concretely.

3066 When n is too small. Consider the following sentences:

- 3067 (6.3) Gorillas always like to groom their friends.
 3068 (6.4) The computer that's on the 3rd floor of our office building crashed.

3069 In each example, the bolded words depend on each other: the likelihood of their
 3070 depends on knowing that gorillas is plural, and the likelihood of crashed depends on
 3071 knowing that the subject is a computer. If the n -grams are not big enough to capture
 3072 this context, then the resulting language model would offer probabilities that are too
 3073 low for these sentences, and too high for sentences that fail basic linguistic tests like
 3074 number agreement.

3075 When n is too big. In this case, it is hard to get good estimates of the n -gram parameters from
 3076 our dataset, because of data sparsity. To handle the gorilla example, it is necessary to
 3077 model 6-grams, which means accounting for V^6 events. Under a very small vocabulary
 3078 of $V = 10^4$, this means estimating the probability of 10^{24} distinct events.

3079 These two problems point to another bias-variance tradeoff (see § 2.1.4). A small
 3080 n -gram size introduces high bias, and a large n -gram size introduces high variance. But

3081 in reality we often have both problems at the same time! Language is full of long-range
 3082 dependencies that we cannot capture because n is too small; at the same time, language
 3083 datasets are full of rare phenomena, whose probabilities we fail to estimate accurately
 3084 because n is too large. One solution is to try to keep n large, while still making low-variance
 3085 estimates of the underlying parameters. To do this, we will introduce a different sort of
 3086 bias: smoothing.

3087 **6.2 Smoothing and discounting**

3088 Limited data is a persistent problem in estimating language models. In § 6.1, we presented
 3089 n -grams as a partial solution. sparse data can be a problem even for low-order n -grams;
 3090 at the same time, many linguistic phenomena, like subject-verb agreement, cannot be
 3091 incorporated into language models without high-order n -grams. It is therefore necessary
 3092 to add additional inductive biases to n -gram language models. This section covers some of
 3093 the most intuitive and common approaches, but there are many more (Chen and Goodman,
 3094 1999).

3095 **6.2.1 Smoothing**

3096 A major concern in language modeling is to avoid the situation $p(\mathbf{w}) = 0$, which could
 3097 arise as a result of a single unseen n-gram. A similar problem arose in Naïve Bayes, and
 3098 the solution was smoothing: adding imaginary “pseudo” counts. The same idea can be
 3099 applied to n -gram language models, as shown here in the bigram case,

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

3100 This basic framework is called Lidstone smoothing, but special cases have other names:

- 3101 • Laplace smoothing corresponds to the case $\alpha = 1$.
 3102 • Jeffreys-Perks law corresponds to the case $\alpha = 0.5$. Manning and Schütze (1999)
 3103 offer more insight on the justifications for this setting.

3104 To maintain normalization, anything that we add to the numerator (α) must also appear
 3105 in the denominator ($V\alpha$). This idea is reflected in the concept of effective counts:

$$c_i^* = (c_i + \alpha) \frac{M}{M + V\alpha}, \quad [6.14]$$

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 $\sum_{i=1}^V c_i^* = \sum_{i=1}^V c_i = M$.

	counts	unsmoothed probability	Lidstone smoothing, $\alpha = 0.1$		Discounting, $d = 0.1$	
			effective counts	smoothed probability	effective counts	smoothed probability
impropriety	8	0.4	7.826	0.391	7.9	0.395
offense	5	0.25	4.928	0.246	4.9	0.245
damage	4	0.2	3.961	0.198	3.9	0.195
deficiencies	2	0.1	2.029	0.101	1.9	0.095
outbreak	1	0.05	1.063	0.053	0.9	0.045
infirmity	0	0	0.097	0.005	0.25	0.013
cephalopods	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, $\underline{}$), 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.

The discount for each n-gram is then computed as,

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

3106 6.2.2 Discounting and backoff

3107 Discounting “borrows” probability mass from observed n -grams and redistributes it. In
 3108 Lidstone smoothing, the borrowing is done by increasing the denominator of the relative
 3109 frequency estimates. The borrowed probability mass is then redistributed by increasing
 3110 the numerator for all n -grams. Another approach would be to borrow the same amount of
 3111 probability mass from all observed n -grams, and redistribute it among only the unobserved
 3112 n -grams. This is called absolute discounting. For example, suppose we set an absolute
 3113 discount $d = 0.1$ in a bigram model, and then redistribute this probability mass equally
 3114 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,

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

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

3115 The term $\alpha(j)$ indicates the amount of probability mass that has been discounted for
 3116 context j . This probability mass is then divided across all the unseen events, $\{i' : c(i', j) =$
 3117 $0\}$, proportional to the unigram probability of each word i' . The discount parameter d can
 3118 be optimized to maximize performance (typically held-out log-likelihood) on a development
 3119 set.

3120 6.2.3 *Interpolation

3121 Backoff is one way to combine different order n -gram models. An alternative approach
 3122 is interpolation: setting the probability of a word in context to a weighted sum of its
 3123 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,

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

3124 In this equation, p_n^* is the unsmoothed empirical probability given by an n -gram language
 3125 model, and λ_n is the weight assigned to this model. To ensure that the interpolated
 3126 $p(\mathbf{w})$ is still a valid probability distribution, the values of λ must obey the constraint,
 3127 $\sum_{n=1}^{n_{\max}} \lambda_n = 1$. But how to find the specific values?

3128 An elegant solution is expectation maximization. Recall from chapter 5 that we can
 3129 think about EM as learning with missing data: we just need to choose missing data such
 3130 that learning would be easy if it weren't missing. What's missing in this case? Think of
 3131 each word w_m as drawn from an n -gram of unknown size, $z_m \in \{1 \dots n_{\max}\}$. This z_m is
 3132 the missing data that we are looking for. Therefore, the application of EM to this problem
 3133 involves the following generative process:

3134 for Each token $w_m, m = 1, 2, \dots, M$ do:
 3135 draw the n -gram size $z_m \sim \text{Categorical}(\lambda)$;
 3136 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,

$$\lambda_z = \frac{\text{count}(Z_m = z)}{M} \tag{6.17}$$

$$\propto \sum_{m=1}^M \delta(Z_m = z). \tag{6.18}$$

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 ,

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

3138 A solution is obtained by iterating between updates to q and λ . The complete algorithm
3139 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                                 $\triangleright$  Initialization
3:      $\lambda_z \leftarrow \frac{1}{n_{\max}}$ 
4:   repeat
5:     for  $m \in \{1, 2, \dots, M\}$  do                                 $\triangleright$  E-step
6:       for  $z \in \{1, 2, \dots, n_{\max}\}$  do
7:          $q_m(z) \leftarrow p_z^*(w_m \mid \mathbf{w}_{1:m-}) \times \lambda_z$ 
8:          $\mathbf{q}_m \leftarrow \text{Normalize}(\mathbf{q}_m)$ 
9:       for  $z \in \{1, 2, \dots, n_{\max}\}$  do                                 $\triangleright$  M-step
10:       $\lambda_z \leftarrow \frac{1}{M} \sum_{m=1}^M q_m(z)$ 
11:   until tired
12:   return  $\boldsymbol{\lambda}$ 

```

3140 6.2.4 *Kneser-Ney smoothing

3141 Kneser-Ney smoothing is based on absolute discounting, but it redistributes the resulting
3142 probability mass in a different way from Katz backoff. Empirical evidence points to
3143 Kneser-Ney smoothing as the state-of-art for n -gram language modeling (Goodman, 2001).
3144 To motivate Kneser-Ney smoothing, consider the example: I recently visited ___. Which of
3145 the following is more likely?

3146 • Francisco

3147 • Duluth

3148 Now suppose that both bigrams visited Duluth and visited Francisco are unobserved
 3149 in the training data, and furthermore, the unigram probability $p_1^*(\text{Francisco})$ is greater than
 3150 $p^*(\text{Duluth})$. Nonetheless we would still guess that $p(\text{visited Duluth}) > p(\text{visited Francisco})$,
 3151 because Duluth is a more “versatile” word: it can occur in many contexts, while Francisco
 3152 usually occurs in a single context, following the word San. This notion of versatility is the
 3153 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,

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

3154 This is the amount of probability mass left to account for versatility, which we define
 3155 via the continuation probability $p_{\text{continuation}}(w)$ as proportional to the number of observed
 3156 contexts in which w appears. The numerator of the continuation probability is the number
 3157 of contexts u in which w appears; the denominator normalizes the probability by summing
 3158 the same quantity over all words w' .

3159 The idea of modeling versatility by counting contexts may seem heuristic, but there is
 3160 an elegant theoretical justification from Bayesian nonparametrics (Teh, 2006). Kneser-Ney
 3161 smoothing on n -grams was the dominant language modeling technique before the arrival
 3162 of neural language models.

3163 6.3 Recurrent neural network language models

3164 N -gram language models have been largely supplanted by neural networks. These models
 3165 do not make the n -gram assumption of restricted context; indeed, they can incorporate
 3166 arbitrarily distant contextual information, while remaining computationally and statistically
 3167 tractable.

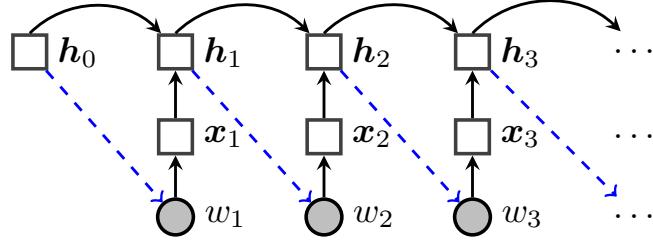


Figure 6.1: The recurrent neural network language model, viewed as an “unrolled” computation graph. Solid lines indicate direct computation, dotted blue lines indicate probabilistic dependencies, circles indicate random variables, and squares indicate computation nodes.

3168 The first insight behind neural language models is to treat word prediction as a discriminative
 3169 learning task.² The goal is to compute the probability $p(w | u)$, where $w \in \mathcal{V}$ is a word, and
 3170 u is the context, which depends on the previous words. Rather than directly estimating
 3171 the word probabilities from (smoothed) relative frequencies, we can treat language
 3172 modeling as a machine learning problem, and estimate parameters that maximize the log
 3173 conditional probability of a corpus.

3174 The second insight is to reparametrize the probability distribution $p(w | u)$ as a function
 3175 of two dense K -dimensional numerical vectors, $\beta_w \in \mathbb{R}^K$, and $v_u \in \mathbb{R}^K$,

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

3176 where $\beta_w \cdot v_u$ represents a dot product. As usual, the denominator ensures that the
 3177 probability distribution is properly normalized. This vector of probabilities is equivalent
 3178 to 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 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 h_m represent the contextual information at position m

²This idea predates neural language models (e.g., Rosenfeld, 1996; Roark et al., 2007).

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 non-linearities (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).

3205 6.3.1 Backpropagation through time

3206 The recurrent neural network language model has the following parameters:

- 3207 • $\phi_i \in \mathbb{R}^K$, the “input” word vectors (these are sometimes called word embeddings,
 3208 since each word is embedded in a K -dimensional space);
 3209 • $\beta_i \in \mathbb{R}^K$, the “output” word vectors;
 3210 • $\Theta \in \mathbb{R}^{K \times K}$, the recurrence operator;
 3211 • \mathbf{h}_0 , the initial state.

3212 Each of these parameters can be estimated by formulating an objective function over the
 3213 training corpus, $L(\mathbf{w})$, and then applying backpropagation to obtain gradients on the
 3214 parameters from a minibatch of training examples (see § 3.3.1). Gradient-based updates
 3215 can be computed from an online learning algorithm such as stochastic gradient descent
 3216 (see § 2.5.2).

3217 The application of backpropagation to recurrent neural networks is known as backpropagation
 3218 through time, because the gradients on units at time m depend in turn on the gradients of
 3219 units at earlier times $n < m$. Let ℓ_{m+1} represent the negative log-likelihood 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 :

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

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

3220 where g' is the local derivative of the nonlinear function g . The key point in this equation
 3221 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
 3222 so on, until reaching the initial state \mathbf{h}_0 .

3223 Each derivative $\frac{\partial \mathbf{h}_m}{\partial \theta_{k,k'}}$ will be reused many times: it appears in backpropagation from
 3224 the loss ℓ_m , but also in all subsequent losses $\ell_{n>m}$. Neural network toolkits such as
 3225 Torch (Collobert et al., 2011) and DyNet (Neubig et al., 2017) compute the necessary

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

6.3.2 Hyperparameters

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

6.3.3 Gated recurrent neural networks

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

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

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

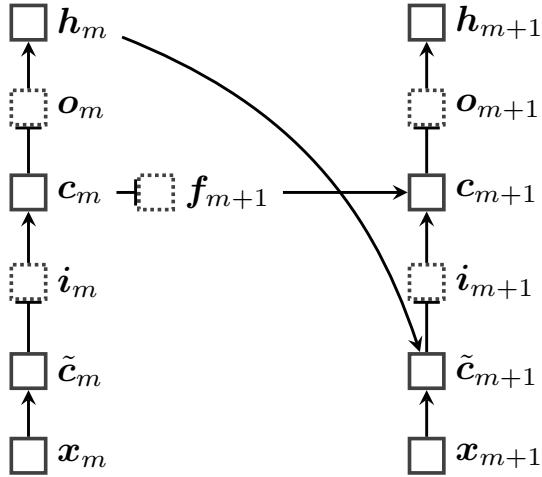


Figure 6.2: The long short-term memory (LSTM) architecture. Gates are shown in boxes with dotted edges. In an LSTM language model, each h_m would be used to predict the next word w_{m+1} .

The gates are functions of the input and previous hidden state. They are computed from elementwise sigmoid activations, $\sigma(x) = (1 + \exp(-x))^{-1}$, ensuring that their values will be in the range $[0, 1]$. They can therefore be viewed as soft, differentiable logic gates. The LSTM architecture is shown in Figure 6.2, and the complete update equations are:

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

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

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

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

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

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

3261 The operator \odot is an elementwise (Hadamard) product. Each gate is controlled by a vector
 3262 of weights, which parametrize the previous hidden state (e.g., $\Theta^{(h \rightarrow f)}$) and the current
 3263 input (e.g., $\Theta^{(x \rightarrow f)}$), plus a vector offset (e.g., b_f). The overall operation can be informally
 3264 summarized as $(h_m, c_m) = \text{LSTM}(x_m, (h_{m-1}, c_{m-1}))$, with (h_m, c_m) representing the
 3265 LSTM state after reading token m .

3266 The LSTM outperforms standard recurrent neural networks across a wide range of
 3267 problems. It was first used for language modeling by Sundermeyer et al. (2012), but can
 3268 be applied more generally: the vector h_m can be treated as a complete representation of

3269 the input sequence up to position m , and can be used for any labeling task on a sequence
 3270 of tokens, as we will see in the next chapter.

3271 There are several LSTM variants, of which the Gated Recurrent Unit (Cho et al., 2014)
 3272 is one of the more well known. Many software packages implement a variety of RNN
 3273 architectures, so choosing between them is simple from a user’s perspective. Jozefowicz
 3274 et al. (2015) provide an empirical comparison of various modeling choices circa 2015.

3275 6.4 Evaluating language models

3276 Language modeling is not usually an application in itself: language models are typically
 3277 components of larger systems, and they would ideally be evaluated extrinsically. This
 3278 means evaluating whether the language model improves performance on the application
 3279 task, such as machine translation or speech recognition. But this is often hard to do, and
 3280 depends on details of the overall system which may be irrelevant to language modeling. In
 3281 contrast, intrinsic evaluation is task-neutral. Better performance on intrinsic metrics may
 3282 be expected to improve extrinsic metrics across a variety of tasks, but there is always the
 3283 risk of over-optimizing the intrinsic metric. This section discusses some intrinsic metrics,
 3284 but keep in mind the importance of performing extrinsic evaluations to ensure that intrinsic
 3285 performance gains carry over to the applications that we care about.

3286 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,

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

3287 treating the entire held-out corpus as a single stream of tokens.

3288 Typically, unknown words are mapped to the $\langle \text{unk} \rangle$ token. This means that we have
 3289 to estimate some probability for $\langle \text{unk} \rangle$ on the training data. One way to do this is to fix
 3290 the vocabulary \mathcal{V} to the $V - 1$ words with the highest counts in the training data, and
 3291 then convert all other tokens to $\langle \text{unk} \rangle$. Other strategies for dealing with out-of-vocabulary
 3292 terms are discussed in § 6.5.

3293 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,

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

3294 where M is the total number of tokens in the held-out corpus.

3295 Lower perplexities correspond to higher likelihoods, so lower scores are better on this
3296 metric — it is better to be less perplexed. Here are some special cases:

- 3297 • In the limit of a perfect language model, probability 1 is assigned to the held-out
3298 corpus, with $\text{Perplex}(\mathbf{w}) = 2^{-\frac{1}{M} \log_2 1} = 2^0 = 1$.
- 3299 • In the opposite limit, probability zero is assigned to the held-out corpus, which
3300 corresponds to an infinite perplexity, $\text{Perplex}(\mathbf{w}) = 2^{-\frac{1}{M} \log_2 0} = 2^\infty = \infty$.
- 3299 • Assume a uniform, unigram model in which $p(w_i) = \frac{1}{V}$ for all words in the vocabulary.
Then,

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

3301 This is the “worst reasonable case” scenario, since you could build such a language
3302 model without even looking at the data.

3303 In practice, language models tend to give perplexities in the range between 1 and V . A
3304 small benchmark dataset is the Penn Treebank, which contains roughly a million tokens; its
3305 vocabulary is limited to 10,000 words, with all other tokens mapped a special `<unk>` symbol.
3306 On this dataset, a well-smoothed 5-gram model achieves a perplexity of 141 (Mikolov and
3307 Zweig, Mikolov and Zweig), and an LSTM language model achieves perplexity of roughly
3308 80 (Zaremba, Sutskever, and Vinyals, Zaremba et al.). Various enhancements to the LSTM
3309 architecture can bring the perplexity below 60 (Merity et al., 2018). A larger-scale language
3310 modeling dataset is the 1B Word Benchmark (Chelba et al., 2013), which contains text
3311 from Wikipedia. On this dataset, a perplexities of around 25 can be obtained by averaging
3312 together multiple LSTM language models (Jozefowicz et al., 2016).

3313 6.5 Out-of-vocabulary words

3314 So far, we have assumed a closed-vocabulary setting — the vocabulary \mathcal{V} is assumed to be
 3315 a finite set. In realistic application scenarios, this assumption may not hold. Consider, for
 3316 example, the problem of translating newspaper articles. The following sentence appeared
 3317 in a Reuters article on January 6, 2017:⁵

3318 The report said U.S. intelligence agencies believe Russian military intelligence,
 3319 the GRU, used intermediaries such as WikiLeaks, DCLeaks.com and the Guccifer
 3320 2.0 "persona" to release emails...

3321 Suppose that you trained a language model on the Gigaword corpus,⁶ which was released
 3322 in 2003. The bolded terms either did not exist at this date, or were not widely known; they
 3323 are unlikely to be in the vocabulary. The same problem can occur for a variety of other
 3324 terms: new technologies, previously unknown individuals, new words (e.g., hashtag), and
 3325 numbers.

3326 One solution is to simply mark all such terms with a special token, $\langle \text{unk} \rangle$. While
 3327 training the language model, we decide in advance on the vocabulary (often the K most
 3328 common terms), and mark all other terms in the training data as $\langle \text{unk} \rangle$. If we do not want
 3329 to determine the vocabulary size in advance, an alternative approach is to simply mark the
 3330 first occurrence of each word type as $\langle \text{unk} \rangle$.

3331 But it is often better to make distinctions about the likelihood of various unknown words.
 3332 This is particularly important in languages that have rich morphological systems, with
 3333 many inflections for each word. For example, Portuguese is only moderately complex from
 3334 a morphological perspective, yet each verb has dozens of inflected forms (see Figure 4.3b).
 3335 In such languages, there will be many word types that we do not encounter in a corpus,
 3336 which are nonetheless predictable from the morphological rules of the language. To use a
 3337 somewhat contrived English example, if transfenestrate is in the vocabulary, our language
 3338 model should assign a non-zero probability to the past tense transfenestrated, even if it
 3339 does not appear in the training data.

3340 One way to accomplish this is to supplement word-level language models with character-level
 3341 language models. Such models can use n -grams or RNNs, but with a fixed vocabulary equal
 3342 to the set of ASCII or Unicode characters. For example Ling et al. (2015) propose an LSTM
 3343 model over characters, and Kim (2014) employ a convolutional neural network (LeCun
 3344 and Bengio, 1995). A more linguistically motivated approach is to segment words into
 3345 meaningful subword units, known as morphemes (see chapter 9). For example, Botha

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>

3346 and Blunsom (2014) induce vector representations for morphemes, which they build into
 3347 a log-bilinear language model; Bhatia et al. (2016) incorporate morpheme vectors into an
 3348 LSTM.

3349 Additional resources

3350 A variety of neural network architectures have been applied to language modeling. Notable
 3351 earlier non-recurrent architectures include the neural probabilistic language model (Bengio
 3352 et al., 2003) and the log-bilinear language model (Mnih and Hinton, 2007). Much more
 3353 detail on these models can be found in the text by Goodfellow et al. (2016).

3354 Exercises

- 3355 1. Prove that n -gram language models give valid probabilities if the n -gram probabilities
 3356 are valid. Specifically, assume that,

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

3357 for all contexts $(w_{m-1}, w_{m-2}, \dots, w_{m-n+1})$. Prove that $\sum_{\mathbf{w}} p_n(\mathbf{w}) = 1$ for all $\mathbf{w} \in \mathcal{V}^*$,
 3358 where p_n is the probability under an n -gram language model. Your proof should
 3359 proceed by induction. You should handle the start-of-string case $p(w_1 | \underbrace{\square, \dots, \square}_{n-1})$,
 3360 but you need not handle the end-of-string token.

- 3361 2. First, show that RNN language models are valid using a similar proof technique to
 3362 the one in the previous problem.

3363 Next, let $p_r(\mathbf{w})$ indicate the probability of \mathbf{w} under RNN r . An ensemble of RNN
 3364 language models computes the probability,

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

3365 Does an ensemble of RNN language models compute a valid probability?

- 3366 3. Consider a unigram language model over a vocabulary of size V . Suppose that a
 3367 word appears m times in a corpus with M tokens in total. With Lidstone smoothing
 3368 of α , for what values of m is the smoothed probability greater than the unsmoothed
 3369 probability?

- 3370 4. Consider a simple language in which each token is drawn from the vocabulary \mathcal{V} with
 3371 probability $\frac{1}{V}$, independent of all other tokens.

3372 Given a corpus of size M , what is the expectation of the fraction of all possible
 3373 bigrams that have zero count? You may assume V is large enough that $\frac{1}{V} \approx \frac{1}{V-1}$.

- 3374 5. Continuing the previous problem, determine the value of M such that the fraction
 3375 of bigrams with zero count is at most $\epsilon \in (0, 1)$. As a hint, you may use the
 3376 approximation $\ln(1 + \alpha) \approx \alpha$ for $\alpha \approx 0$.

- 3377 6. In real languages, word probabilities are neither uniform nor independent.

- 3378 • Assume that word probabilities are independent but not uniform, so that in
 3379 general $p(w) \neq \frac{1}{V}$. Prove that the expected fraction of unseen bigrams will be
 3380 higher than in the IID case.
- 3381 • Assume that word probabilities are neither independent nor uniform. Again,
 3382 prove that the expected fraction of unseen bigrams will be higher than in the
 3383 IID case. [todo: double check]

- 3384 7. Consider a recurrent neural network with a single hidden unit and a sigmoid activation,
 3385 $h_m = \sigma(\theta h_{m-1} + x_m)$. Prove that if $|\theta| < 1$, then the gradient $\frac{\partial h_m}{\partial h_{m-k}}$ goes to zero as
 3386 $k \rightarrow \infty$.⁷

- 3387 8. Zipf's law states that if the word types in a corpus are sorted by frequency, then the
 3388 frequency of the word at rank r is proportional to r^{-s} , where s is a free parameter,
 3389 usually around 1. (Another way to view Zipf's law is that a plot of log frequency
 3390 against log rank will be linear.) Solve for s using the counts of the first and second
 3391 most frequent words, c_1 and c_2 .

- 3392 9. Download the wikitext-2 dataset.⁸ Read and tokenize the training data, e.g. with
 3393 the CountVectorizer class from scikit-learn. Estimate the Zipf's law coefficient by,

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

3394 where $\mathbf{r} = [1, 2, 3, \dots]$ is the vector of ranks of all words in the corpus, and $\mathbf{c} =$
 3395 $[c_1, c_2, c_3, \dots]$ is the vector of counts of all words in the corpus, sorted in descending
 3396 order.

3397 Make a log-log plot of the observed counts, and the expected counts according to
 3398 Zipf's law. The sum $\sum_{r=1}^{\infty} r^s = \zeta(s)$ is the Riemann zeta function, available in
 3399 python's scipy library as `scipy.special.zeta`.

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

⁸Currently available at https://github.com/pytorch/examples/tree/master/word_language_model/data/wikitext-2

- 3400 10. Using the Pytorch library, train an LSTM language model from the Wikitext training
3401 corpus. After each epoch of training, compute its perplexity on the Wikitext validation
3402 corpus. For this exercise, you can focus on the 10^4 most frequent words in the training
3403 corpus, and label everything else as `<unk>`. Stop training when the perplexity stops
3404 improving.

3405 Chapter 7

3406 Sequence labeling

3407 The goal of sequence labeling is to assign tags to words, or more generally, to assign
3408 discrete labels to discrete elements in a sequence. There are many applications of sequence
3409 labeling in natural language processing, and chapter 8 presents an overview. For now,
3410 we'll focus on the classic problem of part-of-speech tagging, which requires tagging each
3411 word by its grammatical category. Coarse-grained grammatical categories include Nouns,
3412 which describe things, properties, or ideas, and Verbs, which describe actions and events.
3413 Consider a simple input:

3414 (7.1) They can fish.

3415 A dictionary of coarse-grained part-of-speech tags might include noun as the only valid tag
3416 for they, but both noun and verb as potential tags for can and fish. An accurate sequence
3417 labeling algorithm should select the verb tag for both can and fish in (7.1), but it should
3418 select noun for the same two words in the phrase can of fish.

3419 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:

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

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

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

3420 Here the feature function takes three arguments as input: the sentence to be tagged (e.g.,
3421 they can fish), the proposed tag (e.g., N or V), and the index of the token to which this

3422 tag is applied. This simple feature function then returns a single feature: a tuple including
 3423 the word to be tagged and the tag that has been proposed. If the vocabulary size is
 3424 V and the number of tags is K , then there are $V \times K$ features. Each of these features
 3425 must be assigned a weight. These weights can be learned from a labeled dataset using a
 3426 classification algorithm such as perceptron, but this isn't necessary in this case: it would
 3427 be equivalent to define the classification weights directly, with $\theta_{w,y} = 1$ for the tag y most
 3428 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:

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

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

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

3429 These features contain enough information that a tagger should be able to choose the
 3430 right tag for the word fish: words that come after can are likely to be verbs, so the feature
 3431 $(w_{m-1} = \text{can}, y_m = V)$ should have a large positive weight.

3432 However, even with this enhanced feature set, it may be difficult to tag some sequences
 3433 correctly. One reason is that there are often relationships between the tags themselves. For
 3434 example, in English it is relatively rare for a verb to follow another verb — particularly
 3435 if we differentiate Modal verbs like can and should from more typical verbs, like give,
 3436 transcend, and befuddle. We would like to incorporate preferences against tag sequences
 3437 like Verb-Verb, and in favor of tag sequences like Noun-Verb. The need for such preferences
 3438 is best illustrated by a garden path sentence:

3439 (7.2) The old man the boat.

3440 Grammatically, the word the is a Determiner. When you read the sentence, what part
 3441 of speech did you first assign to old? Typically, this word is an adjective — abbreviated as
 3442 J — which is a class of words that modify nouns. Similarly, man is usually a noun. The
 3443 resulting sequence of tags is D J N D N. But this is a mistaken “garden path” interpretation,
 3444 which ends up leading nowhere. It is unlikely that a determiner would directly follow a

3445 noun,¹ and it is particularly unlikely that the entire sentence would lack a verb. The
 3446 only possible verb in (7.2) is the word man, which can refer to the act of maintaining
 3447 and piloting something — often boats. But if man is tagged as a verb, then old is seated
 3448 between a determiner and a verb, and must be a noun. And indeed, adjectives often have
 3449 a second interpretation as nouns when used in this way (e.g., the young, the restless). This
 3450 reasoning, in which the labeling decisions are intertwined, cannot be applied in a setting
 3451 where each tag is produced by an independent classification decision.

3452 7.2 Sequence labeling as structure prediction

3453 As an alternative, think of the entire sequence of tags as a label itself. For a given sequence
 3454 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
 3455 $\mathcal{Y} = \{\text{N, V, D, …}\}$ refers to the set of individual tags, and \mathcal{Y}^M refers to the set of tag
 3456 sequences of length M . We can then treat the sequence labeling problem as a classification
 3457 problem in the label space $\mathcal{Y}(\mathbf{w})$,

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}(\mathbf{w})}{\operatorname{argmax}} \Psi(\mathbf{w}, \mathbf{y}), \quad [7.7]$$

3458 where $\mathbf{y} = (y_1, y_2, \dots, y_M)$ is a sequence of M tags, and Ψ is a scoring function on pairs
 3459 of sequences, $V^M \times \mathcal{Y}^M \mapsto \mathbb{R}$. Such a function can include features that capture the
 3460 relationships between tagging decisions, such as the preference that determiners not follow
 3461 nouns, or that all sentences have verbs.

3462 Given that the label space is exponentially large in the length of the sequence M ,
 3463 can it ever be practical to perform tagging in this way? The problem of making a series
 3464 of interconnected labeling decisions is known as inference. Because natural language is
 3465 full of interrelated grammatical structures, inference is a crucial aspect of natural language
 3466 processing. In English, it is not unusual to have sentences of length $M = 20$; part-of-speech
 3467 tag sets vary in size from 10 to several hundred. Taking the low end of this range, we have
 3468 $|\mathcal{Y}(\mathbf{w}_{1:M})| \approx 10^{20}$, one hundred billion billion possible tag sequences. Enumerating and
 3469 scoring each of these sequences would require an amount of work that is exponential in the
 3470 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,

$$\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m), \quad [7.8]$$

3471 where each $\psi(\cdot)$ scores a local part of the tag sequence. Note that the sum goes up to $M+1$,
 3472 so that we can include a score for a special end-of-sequence tag, $\psi(\mathbf{w}_{1:M}, \diamond, y_M, M+1)$.
 3473 We also define a special tag to begin the sequence, $y_0 \triangleq \diamond$.

¹The main exception occurs with ditransitive verbs, such as They gave the winner a trophy.

3474 In a linear model, local scoring function can be defined as a dot product of weights and
 3475 features,

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

3476 The feature vector \mathbf{f} can consider the entire input \mathbf{w} , and can look at pairs of adjacent
 3477 tags. This is a step up from per-token classification: the weights can assign low scores to
 3478 infelicitous tag pairs, such as noun-determiner, and high scores for frequent tag pairs, such
 3479 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):

$$\mathbf{f}(\mathbf{w} = \text{they can fish}, \mathbf{y} = \text{N V V}) = \sum_{m=1}^{M+1} \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.10]$$

$$\begin{aligned} &= \mathbf{f}(\mathbf{w}, \text{N}, \diamond, 1) \\ &\quad + \mathbf{f}(\mathbf{w}, \text{V}, \text{N}, 2) \\ &\quad + \mathbf{f}(\mathbf{w}, \text{V}, \text{V}, 3) \\ &\quad + \mathbf{f}(\mathbf{w}, \blacklozenge, \text{V}, 4) \end{aligned} \quad [7.11]$$

$$\begin{aligned} &= (w_m = \text{they}, y_m = \text{N}) + (y_m = \text{N}, y_{m-1} = \diamond) \\ &\quad + (w_m = \text{can}, y_m = \text{V}) + (y_m = \text{V}, y_{m-1} = \text{N}) \\ &\quad + (w_m = \text{fish}, y_m = \text{V}) + (y_m = \text{V}, y_{m-1} = \text{V}) \\ &\quad + (y_m = \blacklozenge, y_{m-1} = \text{V}). \end{aligned} \quad [7.12]$$

3480 There are seven active features for this example: one for each word-tag pair, and one
 3481 for each tag-tag pair, including a final tag $y_{M+1} = \blacklozenge$. These features capture the two main
 3482 sources of information for part-of-speech tagging in English: which tags are appropriate
 3483 for each word, and which tags tend to follow each other in sequence. Given appropriate
 3484 weights for these features, taggers can achieve high accuracy, even for difficult cases like
 3485 the old man the boat. We will now discuss how this restricted scoring function enables
 3486 efficient inference, through the Viterbi algorithm (Viterbi, 1967).

3487 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:

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} \Psi(\mathbf{w}, \mathbf{y}) \quad [7.13]$$

$$= \operatorname{argmax}_{\mathbf{y}_{1:M}} \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.14]$$

$$= \operatorname{argmax}_{\mathbf{y}_{1:M}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}), \quad [7.15]$$

3488 where the final line simplifies the notation with the shorthand,

$$s_m(y_m, y_{m-1}) \triangleq \psi(\mathbf{w}_{1:M}, y_m, y_{m-1}, m). \quad [7.16]$$

This inference problem can be solved efficiently using dynamic programming, an algorithmic technique for reusing work in recurrent computations. We begin by solving an auxiliary problem: rather than finding the best tag sequence, we compute the score of the best tag sequence,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}, \mathbf{y}_{1:M}) = \max_{\mathbf{y}_{1:M}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}). \quad [7.17]$$

This score involves a maximization over all tag sequences of length M , written $\max_{\mathbf{y}_{1:M}}$. This maximization can be broken into two pieces,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}, \mathbf{y}_{1:M}) = \max_{y_M} \max_{\mathbf{y}_{1:M-1}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}), \quad [7.18]$$

which says that we maximize over the final tag y_M , and we maximize over all “prefixes”, $\mathbf{y}_{1:M-1}$. Within the sum of scores, only the final term $s_{M+1}(\blacklozenge, y_M)$ depends on y_M , so we can pull this term out of the second maximization,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}, \mathbf{y}_{1:M}) = \max_{y_M} s_{M+1}(\blacklozenge, y_M) + \max_{\mathbf{y}_{1:M-1}} \sum_{m=1}^M s_m(y_m, y_{m-1}). \quad [7.19]$$

This same reasoning can be applied recursively to the second term of Equation 7.19, pulling out $s_M(y_M, y_{M-1})$, and so on. We can formalize this idea by defining an auxiliary

Algorithm 11 The Viterbi algorithm. Each $s_m(k, k')$ is a local score for tag $y_m = k$ and $y_{m-1} = k'$.

```

for  $k \in \{0, \dots, K\}$  do
     $v_1(k) = s_1(k, \diamond)$ 
for  $m \in \{2, \dots, M\}$  do
    for  $k \in \{0, \dots, K\}$  do
         $v_m(k) = \max_{k'} s_m(k, k') + v_{m-1}(k')$ 
         $b_m(k) = \operatorname{argmax}_{k'} s_m(k, k') + v_{m-1}(k')$ 
     $y_M = \operatorname{argmax}_k s_{M+1}(\blacklozenge, k) + v_M(k)$ 
    for  $m \in \{M-1, \dots, 1\}$  do
         $y_m = b_m(y_{m+1})$ 
return  $\mathbf{y}_{1:M}$ 
```

variable called the Viterbi variable,

$$v_m(y_m) \triangleq \max_{\mathbf{y}_{1:m-1}} \sum_{n=1}^m s_n(y_n, y_{n-1}) \quad [7.20]$$

$$= \max_{y_{m-1}} s_m(y_m, y_{m-1}) + \max_{\mathbf{y}_{1:m-2}} \sum_{n=1}^{m-1} s_n(y_n, y_{n-1}) \quad [7.21]$$

$$= \max_{y_{m-1}} s_m(y_m, y_{m-1}) + v_{m-1}(y_{m-1}). \quad [7.22]$$

3489 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,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}_{1:M}, \mathbf{y}_{1:M}) = v_{M+1}(\blacklozenge). \quad [7.24]$$

3490 Thus, the score of the best labeling for the sequence can be computed in a single forward
 3491 sweep: first compute all variables $v_1(\cdot)$ from Equation 7.23, and then compute all variables
 3492 $v_2(\cdot)$ from the recurrence in Equation 7.22, continuing until the final variable $v_{M+1}(\blacklozenge)$.

3493 The Viterbi variables can be arranged in a structure known as a trellis, shown in
 3494 Figure 7.1. Each column indexes a token m in the sequence, and each row indexes a tag
 3495 in \mathcal{Y} ; every $v_{m-1}(k)$ is connected to every $v_m(k')$, indicating that $v_m(k')$ is computed from
 3496 $v_{m-1}(k)$. Special nodes are set aside for the start and end states.

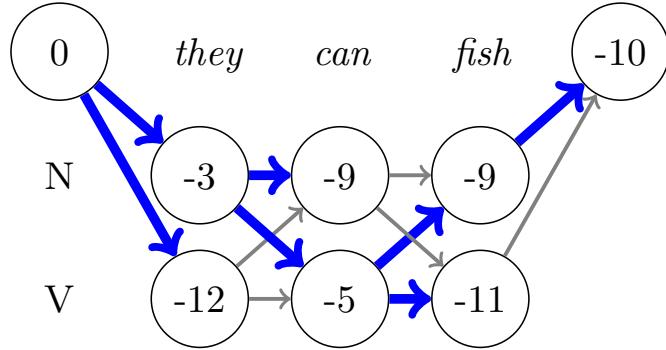


Figure 7.1: The trellis representation of the Viterbi variables, for the example they can fish, using the weights shown in Table 7.1.

3497 Our real goal is to find the best scoring sequence, not simply to compute its score.
 3498 But solving the auxiliary problem gets us almost all the way there. Recall that each $v_m(k)$
 3499 represents the score of the best tag sequence ending in that tag k in position m . To compute
 3500 this, we maximize over possible values of y_{m-1} . If we keep track of the “argmax” tag that
 3501 maximizes this choice at each step, then we can walk backwards from the final tag, and
 3502 recover the optimal tag sequence. This is indicated in Figure 7.1 by the solid blue lines,
 3503 which we trace back from the final position. These backward pointers are written $b_m(k)$,
 3504 indicating the optimal tag y_{m-1} on the path to $Y_m = k$.

3505 The complete Viterbi algorithm is shown in Algorithm 11. When computing the
 3506 initial Viterbi variables $v_1(\cdot)$, the special tag \diamond indicates the start of the sequence. When
 3507 computing the final tag Y_M , another special tag, \blacklozenge indicates the end of the sequence. These
 3508 special tags enable the use of transition features for the tags that begin and end the
 3509 sequence: for example, conjunctions are unlikely to end sentences in English, so we would
 3510 like a low score for $s_{M+1}(\blacklozenge, CC)$; nouns are relatively likely to appear at the beginning of
 3511 sentences, so we would like a high score for $s_1(N, \diamond)$, assuming the noun tag is compatible
 3512 with the first word token w_1 .

3513 Complexity If there are K tags and M positions in the sequence, then there are $M \times K$
 3514 Viterbi variables to compute. Computing each variable requires finding a maximum over K
 3515 possible predecessor tags. The total time complexity of populating the trellis is therefore
 3516 $\mathcal{O}(MK^2)$, with an additional factor for the number of active features at each position.
 3517 After completing the trellis, we simply trace the backwards pointers to the beginning of
 3518 the sequence, which takes $\mathcal{O}(M)$ operations.

they can fish		
N	-2	-3
V	-10	-1

(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 \spadesuit are implicitly set to $-\infty$.

3519 7.3.1 Example

3520 Consider the minimal tagset $\{N, V\}$, corresponding to nouns and verbs. Even in this tagset,
 3521 there is considerable ambiguity: for example, the words can and fish can each take both
 3522 tags. Of the $2 \times 2 \times 2 = 8$ possible taggings for the sentence they can fish, four are possible
 3523 given these possible tags, and two are grammatical.²

3524 The values in the trellis in Figure 7.1 are computed from the feature weights defined in
 3525 Table 7.1. We begin with $v_1(N)$, which has only one possible predecessor, the start tag \diamond .
 3526 This score is therefore equal to $s_1(N, \diamond) = -2 - 1 = -3$, which is the sum of the scores for
 3527 the emission and transition features respectively; the backpointer is $b_1(N) = \diamond$. The score
 3528 for $v_1(V)$ is computed in the same way: $s_1(V, \diamond) = -10 - 2 = -12$, and again $b_1(V) = \diamond$.
 3529 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 $v_4(\spadesuit)$, which is computed as,

$$v_4(\spadesuit) = \max(v_3(N) + s_4(\spadesuit, N), v_3(V) + s_4(\spadesuit, V)) \quad [7.28]$$

$$= \max(-9 + 0 - 1, -11 + 0 - 1) \quad [7.29]$$

$$= -10, \quad [7.30]$$

3530 so $b_4(\spadesuit) = N$. 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.

3531 To compute the optimal tag sequence, we walk backwards from here, next checking
 3532 $b_3(N) = V$, and then $b_2(V) = N$, and finally $b_1(N) = \diamond$. This yields $\mathbf{y} = (N, V, N)$, which
 3533 corresponds to the linguistic interpretation of the fishes being put into cans.

3534 7.3.2 Higher-order features

3535 The Viterbi algorithm was made possible by a restriction of the scoring function to local
 3536 parts that consider only pairs of adjacent tags. We can think of this as a bigram language
 3537 model over tags. A natural question is how to generalize Viterbi to tag trigrams, which
 3538 would involve the following decomposition:

$$\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+2} \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, y_{m-2}, m), \quad [7.31]$$

3539 where $y_{-1} = \diamond$ and $y_{M+2} = \blacklozenge$.

3540 One solution is to create a new tagset $\mathcal{Y}^{(2)}$ from the Cartesian product of the original
 3541 tagset with itself, $\mathcal{Y}^{(2)} = \mathcal{Y} \times \mathcal{Y}$. The tags in this product space are ordered pairs,
 3542 representing adjacent tags at the token level: for example, the tag (N, V) would represent a
 3543 noun followed by a verb. Transitions between such tags must be consistent: we can have a
 3544 transition from (N, V) to (V, N) (corresponding to the tag sequence $N V N$), but not from
 3545 (N, V) to (N, N) , which would not correspond to any coherent tag sequence. This constraint
 3546 can be enforced in feature weights, with $\theta_{((a,b),(c,d))} = -\infty$ if $b \neq c$. The remaining feature
 3547 weights can encode preferences for and against various tag trigrams.

3548 In the Cartesian product tag space, there are K^2 tags, suggesting that the time
 3549 complexity will increase to $\mathcal{O}(MK^4)$. However, it is unnecessary to max over predecessor
 3550 tag bigrams that are incompatible with the current tag bigram. By exploiting this constraint,
 3551 it is possible to limit the time complexity to $\mathcal{O}(MK^3)$. The space complexity grows to
 3552 $\mathcal{O}(MK^2)$, since the trellis must store all possible predecessors of each tag. In general, the
 3553 time and space complexity of higher-order Viterbi grows exponentially with the order of
 3554 the tag n -grams that are considered in the feature decomposition.

3555 7.4 Hidden Markov Models

3556 The Viterbi sequence labeling algorithm is built on the scores $s_m(y, y')$. We will now
 3557 discuss how these scores can be estimated probabilistically. Recall from § 2.1 that the
 3558 probabilistic Naïve Bayes classifier selects the label y to maximize $p(y | \mathbf{x}) \propto p(y, \mathbf{x})$. In
 3559 probabilistic sequence labeling, our goal is similar: select the tag sequence that maximizes
 3560 $p(\mathbf{y} | \mathbf{w}) \propto p(\mathbf{y}, \mathbf{w})$. The locality restriction in Equation 7.8 can be viewed as a conditional
 3561 independence assumption on the random variables \mathbf{y} .

3562 Naïve Bayes was introduced as a generative model — a probabilistic story that explains
 3563 the observed data as well as the hidden label. A similar story can be constructed for

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
  
```

3564 probabilistic sequence labeling: first, the tags are drawn from a prior distribution; next,
 3565 the tokens are drawn from a conditional likelihood. However, for inference to be tractable,
 3566 additional independence assumptions are required. First, the probability of each token
 3567 depends only on its tag, and not on any other element in the sequence:

$$p(\mathbf{w} \mid \mathbf{y}) = \prod_{m=1}^M p(w_m \mid y_m). \quad [7.32]$$

3568 Second, each tag y_m depends only on its predecessor,

$$p(\mathbf{y}) = \prod_{m=1}^M p(y_m \mid y_{m-1}), \quad [7.33]$$

3569 where $y_0 = \diamond$ in all cases. Due to this Markov assumption, probabilistic sequence labeling
 3570 models are known as hidden Markov models (HMMs).

3571 The generative process for the hidden Markov model is shown in Algorithm 12. Given
 3572 the parameters $\boldsymbol{\lambda}$ and $\boldsymbol{\phi}$, we can compute $p(\mathbf{w}, \mathbf{y})$ for any token sequence \mathbf{w} and tag
 3573 sequence \mathbf{y} . The HMM is often represented as a graphical model (Wainwright and Jordan,
 3574 2008), as shown in Figure 7.2. This representation makes the independence assumptions
 3575 explicit: if a variable v_1 is probabilistically conditioned on another variable v_2 , then there
 3576 is an arrow $v_2 \rightarrow v_1$ in the diagram. If there are no arrows between v_1 and v_2 , they are
 3577 conditionally independent, given each variable's Markov blanket. In the hidden Markov
 3578 model, the Markov blanket for each tag y_m includes the “parent” y_{m-1} , and the “children”
 3579 y_{m+1} and w_m .³

3580 It is important to reflect on the implications of the HMM independence assumptions.
 3581 A non-adjacent pair of tags y_m and y_n are conditionally independent; if $m < n$ and we are
 3582 given y_{n-1} , then y_m offers no additional information about y_n . However, if we are not given
 3583 any information about the tags in a sequence, then all tags are probabilistically coupled.

3584 **7.4.1 Estimation**

3585 The hidden Markov model has two groups of parameters:

³In general graphical models, a variable's Markov blanket includes its parents, children, and its children's other parents (Murphy, 2012).

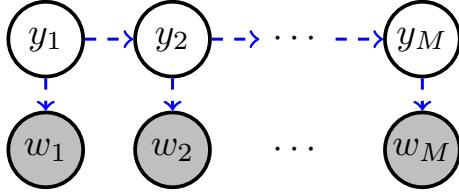


Figure 7.2: Graphical representation of the hidden Markov model. Arrows indicate probabilistic dependencies.

3586 Emission probabilities. The probability $p_e(w_m | y_m; \phi)$ is the emission probability, since
 3587 the words are treated as probabilistically “emitted”, conditioned on the tags.

3588 Transition probabilities. The probability $p_t(y_m | y_{m-1}; \lambda)$ is the transition probability,
 3589 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,

$$\begin{aligned}\phi_{k,i} &\triangleq \Pr(W_m = i | Y_m = k) = \frac{\text{count}(W_m = i, Y_m = k)}{\text{count}(Y_m = k)} \\ \lambda_{k,k'} &\triangleq \Pr(Y_m = k' | Y_{m-1} = k) = \frac{\text{count}(Y_m = k', Y_{m-1} = k)}{\text{count}(Y_{m-1} = k)}.\end{aligned}$$

3590 Smoothing is more important for the emission probability than the transition probability,
 3591 because the vocabulary is much larger than the number of tags.

3592 7.4.2 Inference

3593 The goal of inference in the hidden Markov model is to find the highest probability tag
 3594 sequence,

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}} p(\mathbf{y} | \mathbf{w}). \quad [7.34]$$

3595 As in Naïve Bayes, it is equivalent to find the tag sequence with the highest log-probability,
 3596 since the logarithm is a monotonically increasing function. It is furthermore equivalent to
 3597 maximize the joint probability $p(\mathbf{y}, \mathbf{w}) = p(\mathbf{y} | \mathbf{w}) \times p(\mathbf{w}) \propto p(\mathbf{y} | \mathbf{w})$, which is proportional
 3598 to the conditional probability. Putting these observations together, the inference problem
 3599 can be reformulated as,

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}} \log p(\mathbf{y}, \mathbf{w}). \quad [7.35]$$

We can now apply the HMM independence assumptions:

$$\log p(\mathbf{y}, \mathbf{w}) = \log p(\mathbf{y}) + \log p(\mathbf{w} | \mathbf{y}) \quad [7.36]$$

$$= \sum_{m=1}^{M+1} \log p_Y(y_m | y_{m-1}) + \log p_{W|Y}(w_m | y_m) \quad [7.37]$$

$$= \sum_{m=1}^{M+1} \log \lambda_{y_m, y_{m-1}} + \log \phi_{y_m, w_m} \quad [7.38]$$

$$= \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}), \quad [7.39]$$

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,

$$\phi_{\blacklozenge, w} = \begin{cases} 1, & w = \blacksquare \\ 0, & \text{otherwise,} \end{cases} \quad [7.41]$$

which ensures that the stop tag \blacklozenge 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,

$$s_m(y_m, y_{m-1}) = \log p_y(y_m | y_{m-1}) + \log p_{w|y}(w_m | y_m) \quad [7.42]$$

$$= \log p(y_m, w_m | y_{m-1}). \quad [7.43]$$

Now recall the definition of the Viterbi variables,

$$v_m(y_m) = \max_{y_{m-1}} s_m(y_m, y_{m-1}) + v_{m-1}(y_{m-1}) \quad [7.44]$$

$$= \max_{y_{m-1}} \log p(y_m, w_m | y_{m-1}) + v_{m-1}(y_{m-1}). \quad [7.45]$$

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,

$$v_m(y_m) = \max_{y_{m-1}} \log p(y_m, w_m | y_{m-1}) + \max_{\mathbf{y}_{1:m-2}} \log p(\mathbf{y}_{1:m-1}, \mathbf{w}_{1:m-1}) \quad [7.46]$$

$$= \max_{\mathbf{y}_{1:m-1}} \log p(y_m, w_m | y_{m-1}) + \log p(\mathbf{y}_{1:m-1}, \mathbf{w}_{1:m-1}) \quad [7.47]$$

$$= \max_{\mathbf{y}_{1:m-1}} \log p(\mathbf{y}_{1:m}, \mathbf{w}_{1:m}). \quad [7.48]$$

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 $\mathbf{w}_{1:m}$. 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,

$$v_m(y_m) = \max_{\mathbf{y}_{1:m-1}} p(\mathbf{y}_{1:m-1}, y_m, \mathbf{w}_{1:m}) \quad [7.50]$$

$$= \max_{y_{m-1}} p(y_m, w_m | y_{m-1}) \times \max_{\mathbf{y}_{1:m-2}} p(\mathbf{y}_{1:m-2}, y_{m-1}, \mathbf{w}_{1:m-1}) \quad [7.51]$$

$$= \max_{y_{m-1}} p(y_m, w_m | y_{m-1}) \times v_{m-1}(y_{m-1}) \quad [7.52]$$

$$= p_{w|y}(w_m | y_m) \times \max_{y_{m-1}} p_y(y_m | y_{m-1}) \times v_{m-1}(y_{m-1}). \quad [7.53]$$

3602 Each Viterbi variable is computed by maximizing over a set of products. Thus, the
 3603 Viterbi algorithm is a special case of the max-product algorithm for inference in graphical
 3604 models (Wainwright and Jordan, 2008). However, the product of probabilities tends
 3605 towards zero over long sequences, so the log-probability version of Viterbi is recommended
 3606 in practical implementations.

3607 7.5 Discriminative sequence labeling with features

3608 Today, hidden Markov models are rarely used for supervised sequence labeling. This is
 3609 because HMMs are limited to only two phenomena:

- 3610 • word-tag compatibility, via the emission probability $p_{W|Y}(w_m | y_m)$;
- 3611 • local context, via the transition probability $p_Y(y_m | y_{m-1})$.

3612 The Viterbi algorithm permits the inclusion of richer information in the local scoring
 3613 function $\psi(\mathbf{w}_{1:M}, y_m, y_{m-1}, m)$, which can be defined as a weighted sum of arbitrary local
 3614 features,

$$\psi(\mathbf{w}, y_m, y_{m-1}, m) = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m), \quad [7.54]$$

3615 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:

$$\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.55]$$

$$= \sum_{m=1}^{M+1} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.56]$$

$$= \boldsymbol{\theta} \cdot \sum_{m=1}^{M+1} \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.57]$$

$$= \boldsymbol{\theta} \cdot \mathbf{f}^{(\text{global})}(\mathbf{w}, \mathbf{y}_{1:M}), \quad [7.58]$$

3616 where $\mathbf{f}^{(\text{global})}(\mathbf{w}, \mathbf{y})$ is a global feature vector, which is a sum of local feature vectors,

$$\mathbf{f}^{(\text{global})}(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \mathbf{f}(\mathbf{w}_{1:M}, y_m, y_{m-1}, m), \quad [7.59]$$

3617 with $y_{M+1} = \diamond$ and $y_0 = \diamond$ by construction.

3618 Let's now consider what additional information these features might encode.

3619 Word affix features. Consider the problem of part-of-speech tagging on the first four lines
3620 of the poem Jabberwocky (Carroll, 1917):

3621 (7.3) 'Twas brillig, and the slithy toves
3622 Did gyre and gimble in the wabe:
3623 All mimsy were the borogoves,
3624 And the mome raths outgrabe.

3625 Many of these words were made up by the author of the poem, so a corpus would offer
3626 no information about their probabilities of being associated with any particular part of
3627 speech. Yet it is not so hard to see what their grammatical roles might be in this passage.
3628 Context helps: for example, the word slithy follows the determiner the, so it is probably a
3629 noun or adjective. Which do you think is more likely? The suffix -thy is found in a number
3630 of adjectives, like frothy, healthy, pithy, worthy. It is also found in a handful of nouns —
3631 e.g., apathy, sympathy — but nearly all of these have the longer coda -pathy, unlike slithy.
3632 So the suffix gives some evidence that slithy is an adjective, and indeed it is: later in the
3633 text we find that it is a combination of the adjectives lithe and slimy.⁴

⁴Morphology is the study of how words are formed from smaller linguistic units. chapter 9 touches on computational approaches to morphological analysis. See Bender (2013) for an overview of the underlying linguistic principles, and Haspelmath and Sims (2013) or Lieber (2015) for a full treatment.

3634 Fine-grained context. The hidden Markov model captures contextual information in the
 3635 form of part-of-speech tag bigrams. But sometimes, the necessary contextual information
 3636 is more specific. Consider the noun phrases this fish and these fish. Many part-of-speech
 3637 tagsets distinguish between singular and plural nouns, but do not distinguish between
 3638 singular and plural determiners; for example, the well known Penn Treebank tagset follows
 3639 these conventions. A hidden Markov model would be unable to correctly label fish as
 3640 singular or plural in both of these cases, because it only has access to two features:
 3641 the preceding tag (determiner in both cases) and the word (fish in both cases). The
 3642 classification-based tagger discussed in § 7.1 had the ability to use preceding and succeeding
 3643 words as features, and it can also be incorporated into a Viterbi-based sequence labeler as
 3644 a local feature.

Example. Consider the tagging D J N (determiner, adjective, noun) for the sequence the
 slithy toves, so that

$$\begin{aligned} \mathbf{w} &= \text{the slithy toves} \\ \mathbf{y} &= \text{D J N}. \end{aligned}$$

Let's create the feature vector for this example, assuming that we have word-tag features (indicated by W), tag-tag features (indicated by T), and suffix features (indicated by M). You can assume that you have access to a method for extracting the suffix -thy from slithy, -es from toves, and \emptyset from the, indicating that this word has no suffix.⁵ The resulting feature vector is,

$$\begin{aligned} \mathbf{f}(\text{the slithy toves, D J N}) &= \mathbf{f}(\text{the slithy toves, D}, \diamond, 1) \\ &\quad + \mathbf{f}(\text{the slithy toves, J}, \text{D}, 2) \\ &\quad + \mathbf{f}(\text{the slithy toves, N}, \text{J}, 3) \\ &\quad + \mathbf{f}(\text{the slithy toves}, \blacklozenge, \text{N}, 4) \\ &= \{(T : \diamond, \text{D}), (W : \text{the}, \text{D}), (M : \emptyset, \text{D}), \\ &\quad (T : \text{D}, \text{J}), (W : \text{slithy}, \text{J}), (M : \text{-thy}, \text{J}), \\ &\quad (T : \text{J}, \text{N}), (W : \text{toves}, \text{N}), (M : \text{-es}, \text{N}) \\ &\quad (T : \text{N}, \blacklozenge)\}. \end{aligned}$$

3645 These examples show that local features can incorporate information that lies beyond
 3646 the scope of a hidden Markov model. Because the features are local, it is possible to apply
 3647 the Viterbi algorithm to identify the optimal sequence of tags. The remaining question

⁵Such a system is called a morphological segmenter. The task of morphological segmentation is briefly described in § 9.1.4.4; a well known segmenter is Morfessor (Creutz and Lagus, 2007). In real applications, a typical approach is to include features for all orthographic suffixes up to some maximum number of characters: for slithy, we would have suffix features for -y, -hy, and -thy.

3648 is how to estimate the weights on these features. § 2.2 presented three main types of
 3649 discriminative classifiers: perceptron, support vector machine, and logistic regression. Each
 3650 of these classifiers has a structured equivalent, enabling it to be trained from labeled
 3651 sequences rather than individual tokens.

3652 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:

$$\hat{y} = \operatorname{argmax}_{y \in \mathcal{Y}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y) \quad [7.60]$$

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \mathbf{f}(\mathbf{x}, y) - \mathbf{f}(\mathbf{x}, \hat{y}). \quad [7.61]$$

We can apply exactly the same update in the case of structure prediction,

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \mathbf{y}) \quad [7.62]$$

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \mathbf{f}(\mathbf{w}, \mathbf{y}) - \mathbf{f}(\mathbf{w}, \hat{\mathbf{y}}). \quad [7.63]$$

3653 This learning algorithm is called structured perceptron, because it learns to predict the
 3654 structured output \mathbf{y} . The only difference is that instead of computing $\hat{\mathbf{y}}$ by enumerating the
 3655 entire set \mathcal{Y} , the Viterbi algorithm is used to efficiently search the set of possible taggings,
 3656 \mathcal{Y}^M . Structured perceptron can be applied to other structured outputs as long as efficient
 3657 inference is possible. As in perceptron classification, weight averaging is crucial to get good
 3658 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 $\hat{\mathbf{y}} =$ N V N. Assuming a
 model with features for emissions (w_m, y_m) and transitions (y_{m-1}, y_m) , the corresponding
 structured perceptron update is:

$$\theta_{(\text{fish}, \text{V})} \leftarrow \theta_{(\text{fish}, \text{V})} + 1, \quad \theta_{(\text{fish}, \text{N})} \leftarrow \theta_{(\text{fish}, \text{N})} - 1 \quad [7.64]$$

$$\theta_{(\text{V}, \text{V})} \leftarrow \theta_{(\text{V}, \text{V})} + 1, \quad \theta_{(\text{V}, \text{N})} \leftarrow \theta_{(\text{V}, \text{N})} - 1 \quad [7.65]$$

$$\theta_{(\text{V}, \blacklozenge)} \leftarrow \theta_{(\text{V}, \blacklozenge)} + 1, \quad \theta_{(\text{N}, \blacklozenge)} \leftarrow \theta_{(\text{N}, \blacklozenge)} - 1. \quad [7.66]$$

3659 7.5.2 Structured support vector machines

3660 Large-margin classifiers such as the support vector machine improve on the perceptron by
 3661 pushing the classification boundary away from the training instances. The same idea can be
 3662 applied to sequence labeling. A support vector machine in which the output is a structured

3663 object, such as a sequence, is called a structured support vector machine (Tsochantaridis
 3664 et al., 2004).⁶

3665 In classification, we formalized the large-margin constraint as,

$$\forall y \neq y^{(i)}, \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y^{(i)}) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y) \geq 1, \quad [7.67]$$

3666 requiring a margin of at least 1 between the scores for all labels y that are not equal to the
 3667 correct label $y^{(i)}$. The weights $\boldsymbol{\theta}$ are then learned by constrained optimization (see § 2.3.2).

3668 This idea can be applied to sequence labeling by formulating an equivalent set of
 3669 constraints for all possible labelings $\mathcal{Y}(\mathbf{w})$ for an input \mathbf{w} . However, there are two problems.
 3670 First, in sequence labeling, some predictions are more wrong than others: we may miss only
 3671 one tag out of fifty, or we may get all fifty wrong. We would like our learning algorithm to
 3672 be sensitive to this difference. Second, the number of constraints is equal to the number of
 3673 possible labelings, which is exponentially large in the length of the sequence.

3674 The first problem can be addressed by adjusting the constraint to require larger margins
 3675 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
 3676 true label is $\mathbf{y}^{(i)}$. We can then generalize the margin constraint,

$$\forall \mathbf{y}, \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) \geq c(\mathbf{y}^{(i)}, \mathbf{y}). \quad [7.68]$$

3677 This cost-augmented margin constraint specializes to the constraint in Equation 7.67 if we
 3678 choose the delta function $c(\mathbf{y}^{(i)}, \mathbf{y}) = \delta((\mathbf{y}^{(i)} \neq \mathbf{y}))$. A more expressive cost function is the
 3679 Hamming cost,

$$c(\mathbf{y}^{(i)}, \mathbf{y}) = \sum_{m=1}^M \delta(y_m^{(i)} \neq y_m), \quad [7.69]$$

3680 which computes the number of errors in \mathbf{y} . By incorporating the cost function as the
 3681 margin constraint, we require that the true labeling be separated from the alternatives by
 3682 a margin that is proportional to the number of incorrect tags in each alternative labeling.

The second problem is that the number of constraints is exponential in the length
 of the sequence. This can be addressed by focusing on the prediction $\hat{\mathbf{y}}$ that maximally
 violates the margin constraint. This prediction can be identified by solving the following
 cost-augmented decoding problem:

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \neq \mathbf{y}^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) + c(\mathbf{y}^{(i)}, \mathbf{y}) \quad [7.70]$$

$$= \operatorname{argmax}_{\mathbf{y} \neq \mathbf{y}^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) + c(\mathbf{y}^{(i)}, \mathbf{y}), \quad [7.71]$$

⁶This model is also known as a max-margin Markov network (Taskar et al., 2003), emphasizing that the scoring function is constructed from a sum of components, which are Markov independent.

3683 where in the second line we drop the term $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})$, which is constant in \mathbf{y} .

We can now reformulate the margin constraint for sequence labeling,

$$\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) - \max_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} (\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) + c(\mathbf{y}^{(i)}, \mathbf{y})) \geq 0. \quad [7.72]$$

3684 If the score for $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})$ is greater than the cost-augmented score for all alternatives,
 3685 then the constraint will be met. The name “cost-augmented decoding” is due to the fact
 3686 that the objective includes the standard decoding problem, $\max_{\hat{\mathbf{y}} \in \mathcal{Y}(\mathbf{w})} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \hat{\mathbf{y}})$, plus
 3687 an additional term for the cost. Essentially, we want to train against predictions that are
 3688 strong and wrong: they should score highly according to the model, yet incur a large loss
 3689 with respect to the ground truth. Training adjusts the weights to reduce the score of these
 3690 predictions.

3691 For cost-augmented decoding to be tractable, the cost function must decompose into
 3692 local parts, just as the feature function $\mathbf{f}(\cdot)$ does. The Hamming cost, defined above, obeys
 3693 this property. To perform cost-augmented decoding using the Hamming cost, we need only
 3694 to add features $f_m(y_m) = \delta(y_m \neq y_m^{(i)})$, and assign a constant weight of 1 to these features.
 3695 Decoding can then be performed using the Viterbi algorithm.⁷

As with large-margin classifiers, it is possible to formulate the learning problem in an unconstrained form, by combining a regularization term on the weights and a Lagrangian for the constraints:

$$\min_{\boldsymbol{\theta}} \frac{1}{2} \|\boldsymbol{\theta}\|_2^2 - C \left(\sum_i \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) - \max_{\mathbf{y} \in \mathcal{Y}(\mathbf{w}^{(i)})} [\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) + c(\mathbf{y}^{(i)}, \mathbf{y})] \right), \quad [7.73]$$

3696 In this formulation, C is a parameter that controls the tradeoff between the regularization
 3697 term and the margin constraints. A number of optimization algorithms have been proposed
 3698 for structured support vector machines, some of which are discussed in § 2.3.2. An empirical
 3699 comparison by Kummerfeld et al. (2015) shows that stochastic subgradient descent —
 3700 which is essentially a cost-augmented version of the structured perceptron — is highly
 3701 competitive.

3702 7.5.3 Conditional random fields

3703 The conditional random field (CRF; Lafferty et al., 2001) is a conditional probabilistic
 3704 model for sequence labeling; just as structured perceptron is built on the perceptron

⁷Are there cost functions that do not decompose into local parts? Suppose we want to assign a constant loss c to any prediction $\hat{\mathbf{y}}$ in which k or more predicted tags are incorrect, and zero loss otherwise. This loss function is combinatorial over the predictions, and thus we cannot decompose it into parts.

3705 classifier, conditional random fields are built on the logistic regression classifier.⁸ The
 3706 basic probability model is,

$$p(\mathbf{y} \mid \mathbf{w}) = \frac{\exp(\Psi(\mathbf{w}, \mathbf{y}))}{\sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp(\Psi(\mathbf{w}, \mathbf{y}'))}. \quad [7.74]$$

3707 This is almost identical to logistic regression, but because the label space is now tag
 3708 sequences, we require efficient algorithms for both decoding (searching for the best tag
 3709 sequence given a sequence of words \mathbf{w} and a model $\boldsymbol{\theta}$) and for normalizing (summing over
 3710 all tag sequences). These algorithms will be based on the usual locality assumption on the
 3711 scoring function, $\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m)$.

3712 7.5.3.1 Decoding in CRFs

Decoding — finding the tag sequence $\hat{\mathbf{y}}$ that maximizes $p(\mathbf{y} \mid \mathbf{w})$ — is a direct application of the Viterbi algorithm. The key observation is that the decoding problem does not depend on the denominator of $p(\mathbf{y} \mid \mathbf{w})$,

$$\begin{aligned} \hat{\mathbf{y}} &= \operatorname{argmax}_{\mathbf{y}} \log p(\mathbf{y} \mid \mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{y}} \Psi(\mathbf{y}, \mathbf{w}) - \log \sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp \Psi(\mathbf{y}', \mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{y}} \Psi(\mathbf{y}, \mathbf{w}) = \operatorname{argmax}_{\mathbf{y}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}). \end{aligned}$$

3713 This is identical to the decoding problem for structured perceptron, so the same Viterbi
 3714 recurrence as defined in Equation 7.22 can be used.

3715 7.5.3.2 Learning in CRFs

As with logistic regression, the weights $\boldsymbol{\theta}$ are learned by minimizing the regularized negative log-probability,

$$\ell = \frac{\lambda}{2} \|\boldsymbol{\theta}\|^2 - \sum_{i=1}^N \log p(\mathbf{y}^{(i)} \mid \mathbf{w}^{(i)}; \boldsymbol{\theta}) \quad [7.75]$$

$$= \frac{\lambda}{2} \|\boldsymbol{\theta}\|^2 - \sum_{i=1}^N \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) + \log \sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w}^{(i)})} \exp (\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}')), \quad [7.76]$$

⁸The name “conditional random field” is derived from Markov random fields, a general class of models in which the probability of a configuration of variables is proportional to a product of scores across pairs (or 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.

3716 where λ controls the amount of regularization. The final term in Equation 7.76 is a sum
 3717 over all possible labelings. This term is the log of the denominator in Equation 7.74,
 3718 sometimes known as the partition function.⁹ There are $|\mathcal{Y}|^M$ possible labelings of an input
 3719 of size M , so we must again exploit the decomposition of the scoring function to compute
 3720 this sum efficiently.

The sum $\sum_{\mathbf{y} \in \mathcal{Y}^{w(i)}} \exp \Psi(\mathbf{y}, \mathbf{w})$ can be computed efficiently using the forward recurrence, which is closely related to the Viterbi recurrence. We first define a set of forward variables, $\alpha_m(y_m)$, which is equal to the sum of the scores of all paths leading to tag y_m at position m :

$$\alpha_m(y_m) \triangleq \sum_{\mathbf{y}_{1:m-1}} \exp \sum_{n=1}^m s_n(y_n, y_{n-1}) \quad [7.77]$$

$$= \sum_{\mathbf{y}_{1:m-1}} \prod_{n=1}^m \exp s_n(y_n, y_{n-1}). \quad [7.78]$$

Note the similarity to the definition of the Viterbi variable, $v_m(y_m) = \max_{\mathbf{y}_{1:m-1}} \sum_{n=1}^m s_n(y_n, y_{n-1})$. In the hidden Markov model, the Viterbi recurrence had an alternative interpretation as the max-product algorithm (see Equation 7.53); analogously, the forward recurrence is known as the sum-product algorithm, because of the form of [7.78]. The forward variable can also be computed through a recurrence:

$$\alpha_m(y_m) = \sum_{\mathbf{y}_{1:m-1}} \prod_{n=1}^m \exp s_n(y_n, y_{n-1}) \quad [7.79]$$

$$= \sum_{y_{m-1}} (\exp s_m(y_m, y_{m-1})) \sum_{\mathbf{y}_{1:m-2}} \prod_{n=1}^{m-1} \exp s_n(y_n, y_{n-1}) \quad [7.80]$$

$$= \sum_{y_{m-1}} (\exp s_m(y_m, y_{m-1})) \times \alpha_{m-1}(y_{m-1}). \quad [7.81]$$

Using the forward recurrence, it is possible to compute the denominator of the conditional probability,

$$\sum_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} \Psi(\mathbf{w}, \mathbf{y}) = \sum_{\mathbf{y}_{1:M}} s_{M+1}(\blacklozenge, y_M) \prod_{m=1}^M s_m(y_m, y_{m-1}) \quad [7.82]$$

$$= \alpha_{M+1}(\blacklozenge). \quad [7.83]$$

⁹The terminology of “potentials” and “partition functions” comes from statistical mechanics (Bishop, 2006).

The conditional log-likelihood can be rewritten,

$$\ell = \frac{\lambda}{2} \|\boldsymbol{\theta}\|^2 - \sum_{i=1}^N \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) + \log \alpha_{M+1}(\blacklozenge). \quad [7.84]$$

- 3721 Probabilistic programming environments, such as Torch (Collobert et al., 2011) and dynet (Neubig
 3722 et al., 2017), can compute the gradient of this objective using automatic differentiation.
 3723 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:

$$\frac{d\ell}{d\theta_j} = \lambda \theta_j + \sum_{i=1}^N E[f_j(\mathbf{w}^{(i)}, \mathbf{y})] - f_j(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}), \quad [7.85]$$

- 3724 where $f_j(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})$ refers to the count of feature j for token sequence $\mathbf{w}^{(i)}$ and tag sequence
 3725 $\mathbf{y}^{(i)}$. The expected feature counts are computed “under the hood” when automatic differentiation
 3726 is applied to Equation 7.84 (Eisner, 2016).

3727 Before the widespread use of automatic differentiation, it was common to compute
 3728 the feature expectations from marginal tag probabilities $p(y_m | \mathbf{w})$. These marginal
 3729 probabilities are sometimes useful on their own, and can be computed using the forward-backward
 3730 algorithm. This algorithm combines the forward recurrence with an equivalent backward
 3731 recurrence, which traverses the input from w_M back to w_1 .

3732 7.5.3.3 *Forward-backward algorithm

Marginal probabilities over tag bigrams can be written as,¹⁰

$$\Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}) = \frac{\sum_{\mathbf{y}: Y_m=k, Y_{m-1}=k'} \prod_{n=1}^M \exp s_n(y_n, y_{n-1})}{\sum_{\mathbf{y}'} \prod_{n=1}^M \exp s_n(y'_n, y'_{n-1})}. \quad [7.86]$$

The numerator sums over all tag sequences that include the transition $(Y_{m-1} = k') \rightarrow (Y_m = k)$. Because we are only interested in sequences that include the tag bigram, this sum can be decomposed into three parts: the prefixes $\mathbf{y}_{1:m-1}$, terminating in $Y_{m-1} = k'$; the transition $(Y_{m-1} = k') \rightarrow (Y_m = k)$; and the suffixes $\mathbf{y}_{m:M}$, beginning with the tag

¹⁰Recall the notational convention of upper-case letters for random variables, e.g. Y_m , and lower case letters for specific values, e.g., y_m , so that $Y_m = k$ is interpreted as the event of random variable Y_m taking the value k .

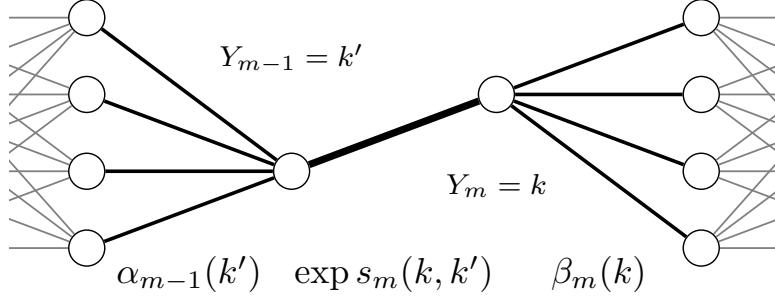


Figure 7.3: A schematic illustration of the computation of the marginal probability $\Pr(Y_{m-1} = k', Y_m = k)$, using the forward score $\alpha_{m-1}(k')$ and the backward score $\beta_m(k)$.

$Y_m = k$:

$$\sum_{\mathbf{y}: Y_m = k, Y_{m-1} = k'} \prod_{n=1}^M \exp s_n(y_n, y_{n-1}) = \sum_{\mathbf{y}_{1:m-1}: Y_{m-1} = k'} \prod_{n=1}^{m-1} \exp s_n(y_n, y_{n-1}) \times \exp s_m(k, k') \times \sum_{\mathbf{y}_{m:M}: Y_m = k} \prod_{n=m+1}^{M+1} \exp s_n(y_n, y_{n-1}). \quad [7.87]$$

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:

$$\beta_m(k) \triangleq \sum_{\mathbf{y}_{m:M}: Y_m = k} \prod_{n=m}^{M+1} \exp s_n(y_n, y_{n-1}) \quad [7.88]$$

$$= \sum_{k' \in \mathcal{Y}} \exp s_{m+1}(k', k) \sum_{\mathbf{y}_{m+1:M}: Y_m = k'} \prod_{n=m+1}^{M+1} \exp s_n(y_n, y_{n-1}) \quad [7.89]$$

$$= \sum_{k' \in \mathcal{Y}} \exp s_{m+1}(k', k) \times \beta_{m+1}(k'). \quad [7.90]$$

3733 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,

$$\log \alpha_m(k) = \log \sum_{k' \in \mathcal{Y}} \exp (\log s_m(k, k') + \log \alpha_{m-1}(k')) \quad [7.91]$$

$$\log \beta_{m-1}(k) = \log \sum_{k' \in \mathcal{Y}} \exp (\log s_m(k', k) + \log \beta_m(k')). \quad [7.92]$$

3734 The application of the forward and backward probabilities is shown in Figure 7.3.
 3735 Both the forward and backward recurrences operate on the trellis, which implies a space
 3736 complexity $\mathcal{O}(MK)$. Because both recurrences require computing a sum over K terms at
 3737 each node in the trellis, their time complexity is $\mathcal{O}(MK^2)$.

3738 7.6 Neural sequence labeling

3739 In neural network approaches to sequence labeling, we construct a vector representation
 3740 for each tagging decision, based on the word and its context. Neural networks can perform
 3741 tagging as a per-token classification decision, or they can be combined with the Viterbi
 3742 algorithm to tag the entire sequence globally.

3743 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,

$$\mathbf{h}_m = g(\mathbf{x}_m, \mathbf{h}_{m-1}), \quad m = 1, 2, \dots, M,$$

3744 where \mathbf{x}_m is the vector embedding of the token w_m and the function g defines the recurrence.
 3745 The starting condition \mathbf{h}_0 is an additional parameter of the model. The long short-term
 3746 memory (LSTM) is a more complex recurrence, in which a memory cell is through a series of
 3747 gates, avoiding repeated application of the non-linearity. Despite these bells and whistles,
 3748 both models share the basic architecture of recurrent updates across a sequence, and both
 3749 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 :

$$\psi_m(y) = \boldsymbol{\beta}_y \cdot \mathbf{h}_m \quad [7.93]$$

$$\hat{y}_m = \underset{y}{\operatorname{argmax}} \psi_m(y). \quad [7.94]$$

3750 The score $\psi_m(y)$ can also be converted into a probability distribution using the usual
 3751 softmax operation,

$$p(y | \mathbf{w}_{1:m}) = \frac{\exp \psi_m(y)}{\sum_{y' \in \mathcal{Y}} \exp \psi_m(y')}. \quad [7.95]$$

3752 Using this transformation, it is possible to train the tagger from the negative log-likelihood
 3753 of the tags, as in a conditional random field. Alternatively, a hinge loss or margin loss
 3754 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:

$$\overleftarrow{\mathbf{h}}_m = g(\mathbf{x}_m, \overleftarrow{\mathbf{h}}_{m+1}), \quad m = 1, 2, \dots, M. \quad [7.96]$$

3755 The hidden states of the left-to-right RNN are denoted $\overrightarrow{\mathbf{h}}_m$. The left-to-right and right-to-left
 3756 vectors are concatenated, $\mathbf{h}_m = [\overleftarrow{\mathbf{h}}_m; \overrightarrow{\mathbf{h}}_m]$. The scoring function in Equation 7.93 is
 3757 applied to this concatenated vector.

3758 Bidirectional RNN tagging has several attractive properties. Ideally, the representation
 3759 \mathbf{h}_m summarizes the useful information from the surrounding context, so that it is not
 3760 necessary to design explicit features to capture this information. If the vector \mathbf{h}_m is an
 3761 adequate summary of this context, then it may not even be necessary to perform the tagging
 3762 jointly: in general, the gains offered by joint tagging of the entire sequence are diminished
 3763 as the individual tagging model becomes more powerful. Using backpropagation, the word
 3764 vectors \mathbf{x} can be trained “end-to-end”, so that they capture word properties that are
 3765 useful for the tagging task. Alternatively, if limited labeled data is available, we can use
 3766 word embeddings that are “pre-trained” from unlabeled data, using a language modeling
 3767 objective (as in § 6.3) or a related word embedding technique (see chapter 14). It is even
 3768 possible to combine both fine-tuned and pre-trained embeddings in a single model.

3769 Neural structure prediction The bidirectional recurrent neural network incorporates information
 3770 from throughout the input, but each tagging decision is made independently. In some
 3771 sequence labeling applications, there are very strong dependencies between tags: it may
 3772 even be impossible for one tag to follow another. In such scenarios, the tagging decision
 3773 must be made jointly across the entire sequence.

3774 Neural sequence labeling can be combined with the Viterbi algorithm by defining the
 3775 local scores as:

$$s_m(y_m, y_{m-1}) = \beta_{y_m} \cdot \mathbf{h}_m + \eta_{y_{m-1}, y_m}, \quad [7.97]$$

3776 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}
 3777 is a scalar parameter for the tag transition (y_{m-1}, y_m) . These local scores can then be
 3778 incorporated into the Viterbi algorithm for inference, and into the forward algorithm for
 3779 training. This model is shown in Figure 7.4. It can be trained from the conditional
 3780 log-likelihood objective defined in Equation 7.76, backpropagating to the tagging parameters

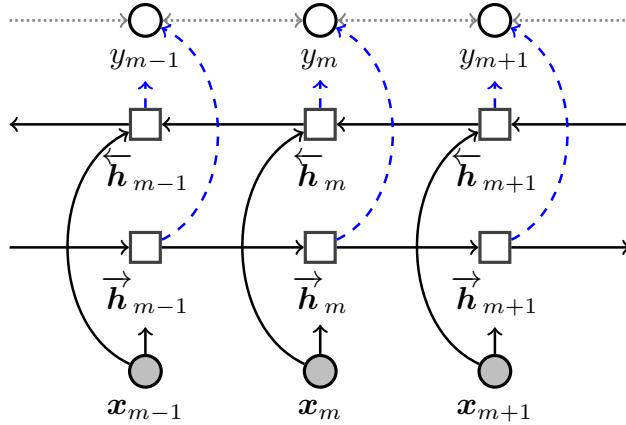


Figure 7.4: Bidirectional LSTM for sequence labeling. The solid lines indicate computation, the dashed lines indicate probabilistic dependency, and the dotted lines indicate the optional additional probabilistic dependencies between labels in the biLSTM-CRF.

3781 β and η , as well as the parameters of the RNN. This model is called the LSTM-CRF, due
 3782 to its combination of aspects of the long short-term memory and conditional random field
 3783 models (Huang et al., 2015).

3784 The LSTM-CRF is especially effective on the task of named entity recognition (Lample
 3785 et al., 2016), a sequence labeling task that is described in detail in § 8.3. This task has
 3786 strong dependencies between adjacent tags, so structure prediction is especially important.

3787 7.6.2 Character-level models

3788 As in language modeling, rare and unseen words are a challenge: if we encounter a word
 3789 that was not in the training data, then there is no obvious choice for the word embedding
 3790 x_m . One solution is to use a generic unseen word embedding for all such words. However,
 3791 in many cases, properties of unseen words can be guessed from their spellings. For example,
 3792 whimsical does not appear in the Universal Dependencies (UD) English Treebank, yet the
 3793 suffix -al makes it likely to be adjective; by the same logic, unflinchingly is likely to be an
 3794 adverb, and barnacle is likely to be a noun.

3795 In feature-based models, these morphological properties were handled by suffix features;
 3796 in a neural network, they can be incorporated by constructing the embeddings of unseen
 3797 words from their spellings or morphology. One way to do this is to incorporate an additional
 3798 layer of bidirectional RNNs, one for each word in the vocabulary (Ling et al., 2015). For
 3799 each such character-RNN, the inputs are the characters, and the output is the concatenation
 3800 of the final states of the left-facing and right-facing passes, $\phi_w = [\vec{h}_{N_w}^{(w)}; \vec{h}_0^{(w)}]$, where $\vec{h}_{N_w}^{(w)}$
 3801 is the final state of the right-facing pass for word w , and N_w is the number of characters

3802 in the word. The character RNN model is trained by backpropagation from the tagging
 3803 objective. On the test data, the trained RNN is applied to out-of-vocabulary words (or all
 3804 words), yielding inputs to the word-level tagging RNN. Other approaches to compositional
 3805 word embeddings are described in § 14.7.1.

3806 7.6.3 Convolutional Neural Networks for Sequence Labeling

3807 One disadvantage of recurrent neural networks is that the architecture requires iterating
 3808 through the sequence of inputs and predictions: each hidden vector \mathbf{h}_m must be computed
 3809 from the previous hidden vector \mathbf{h}_{m-1} , before predicting the tag y_m . These iterative
 3810 computations are difficult to parallelize, and fail to exploit the speedups offered by graphics
 3811 processing units (GPUs) on operations such as matrix multiplication. Convolutional neural
 3812 networks achieve better computational performance by predicting each label y_m from
 3813 a set of matrix operations on the neighboring word embeddings, $\mathbf{x}_{m-k:m+k}$ (Collobert
 3814 et al., 2011). Because there is no hidden state to update, the predictions for each y_m
 3815 can be computed in parallel. For more on convolutional neural networks, see § 3.4.
 3816 Character-based word embeddings can also be computed using convolutional neural networks (Santos
 3817 and Zadrozny, 2014).

3818 7.7 *Unsupervised sequence labeling

3819 In unsupervised sequence labeling, the goal is to induce a hidden Markov model from
 3820 a corpus of unannotated text $(\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \dots, \mathbf{w}^{(N)})$, where each $\mathbf{w}^{(i)}$ is a sequence of
 3821 length $M^{(i)}$. This is an example of the general problem of structure induction, which
 3822 is the unsupervised version of structure prediction. The tags that result from unsupervised
 3823 sequence labeling might be useful for some downstream task, or they might help us to
 3824 better understand the language’s inherent structure. For part-of-speech tagging, it is
 3825 common to use a tag dictionary that lists the allowed tags for each word, simplifying
 3826 the problem (Christodoulopoulos et al., 2010).

Unsupervised learning in hidden Markov models can be performed using the Baum-Welch algorithm, which combines the forward-backward algorithm (§ 7.5.3.3) with expectation-maximization (EM; § 5.1.2). In the M-step, the HMM parameters from expected counts:

$$\Pr(W = i \mid Y = k) = \phi_{k,i} = \frac{E[\text{count}(W = i, Y = k)]}{E[\text{count}(Y = k)]}$$

$$\Pr(Y_m = k \mid Y_{m-1} = k') = \lambda_{k',k} = \frac{E[\text{count}(Y_m = k, Y_{m-1} = k')]}{E[\text{count}(Y_{m-1} = k')]}.$$

3827 The expected counts are computed in the E-step, using the forward and backward
 3828 recurrences. The local scores follow the usual definition for hidden Markov models,

$$s_m(k, k') = \log p_E(w_m | Y_m = k; \boldsymbol{\phi}) + \log p_T(Y_m = k | Y_{m-1} = k'; \lambda). \quad [7.98]$$

The expected transition counts for a single instance are,

$$E[\text{count}(Y_m = k, Y_{m-1} = k') | \boldsymbol{w}] = \sum_{m=1}^M \Pr(Y_{m-1} = k', Y_m = k | \boldsymbol{w}) \quad [7.99]$$

$$= \frac{\sum_{\mathbf{y}: Y_m=k, Y_{m-1}=k'} \prod_{n=1}^M \exp s_n(y_n, y_{n-1})}{\sum_{\mathbf{y}'} \prod_{n=1}^M \exp s_n(y'_n, y'_{n-1})}. \quad [7.100]$$

As described in § 7.5.3.3, these marginal probabilities can be computed from the forward-backward recurrence,

$$\Pr(Y_{m-1} = k', Y_m = k | \boldsymbol{w}) = \frac{\alpha_{m-1}(k') \times s_m(k, k') \times \beta_m(k)}{\alpha_{M+1}(\blacklozenge)}. \quad [7.101]$$

In a hidden Markov model, each element of the forward-backward computation has a special interpretation:

$$\alpha_{m-1}(k') = p(Y_{m-1} = k', \boldsymbol{w}_{1:m-1}) \quad [7.102]$$

$$s_m(k, k') = p(Y_m = k, w_m | Y_{m-1} = k') \quad [7.103]$$

$$\beta_m(k) = p(\boldsymbol{w}_{m+1:M} | Y_m = k). \quad [7.104]$$

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,

$$\begin{aligned} \alpha_{m-1}(k') \times s_m(k, k') \times \beta_m(k) &= p(Y_{m-1} = k', \boldsymbol{w}_{1:m-1}) \\ &\quad \times p(Y_m = k, w_m | Y_{m-1} = k') \\ &\quad \times p(\boldsymbol{w}_{m+1:M} | Y_m = k) \\ &= p(Y_{m-1} = k', Y_m = k, \boldsymbol{w}_{1:M}). \end{aligned} \quad [7.105]$$

Dividing by $\alpha_{M+1}(\blacklozenge) = p(\boldsymbol{w}_{1:M})$ gives the desired probability,

$$\frac{\alpha_{m-1}(k') \times s_m(k, k') \times \beta_m(k)}{\alpha_{M+1}(\blacklozenge)} = \frac{p(Y_{m-1} = k', Y_m = k, \boldsymbol{w}_{1:M})}{p(\boldsymbol{w}_{1:M})} \quad [7.106]$$

$$= \Pr(Y_{m-1} = k', Y_m = k | \boldsymbol{w}_{1:M}). \quad [7.107]$$

3829 The expected emission counts can be computed in a similar manner, using the product
 3830 $\alpha_m(k) \times \beta_m(k)$.

3831 7.7.1 Linear dynamical systems

3832 The forward-backward algorithm can be viewed as Bayesian state estimation in a discrete
 3833 state space. In a continuous state space, $\mathbf{y}_m \in \mathbb{R}^K$, the equivalent algorithm is the Kalman
 3834 smoother. It also computes marginals $p(\mathbf{y}_m | \mathbf{x}_{1:M})$, using a similar two-step algorithm of
 3835 forward and backward passes. Instead of computing a trellis of values at each step, the
 3836 Kalman smoother computes a probability density function $q_{\mathbf{y}_m}(\mathbf{y}_m; \boldsymbol{\mu}_m, \Sigma_m)$, characterized
 3837 by a mean $\boldsymbol{\mu}_m$ and a covariance Σ_m around the latent state. Connections between the
 3838 Kalman Smoother and the forward-backward algorithm are elucidated by Minka (1999)
 3839 and Murphy (2012).

3840 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):

$$p(y_m | \mathbf{y}_{-m}, \mathbf{w}_{1:M}) \propto p(w_m | y_m)p(y_m | \mathbf{y}_{-m}). \quad [7.108]$$

3841 Gibbs Sampling has been applied to unsupervised part-of-speech tagging by Goldwater
 3842 and Griffiths (2007). Beam sampling is a more sophisticated sampling algorithm, which
 3843 randomly draws entire sequences $\mathbf{y}_{1:M}$, rather than individual tags y_m ; this algorithm
 3844 was applied to unsupervised part-of-speech tagging by Van Gael et al. (2009). Spectral
 3845 learning (see § 5.5.2) can also be applied to sequence labeling. By factoring matrices of
 3846 co-occurrence counts of word bigrams and trigrams (Song et al., 2010; Hsu et al., 2012), it
 3847 is possible to obtain globally optimal estimates of the transition and emission parameters,
 3848 under mild assumptions.

3849 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.¹¹ Given these operators, we can denote a generalized

¹¹In a semiring, the addition and multiplication operators must both obey associativity, and multiplication must distribute across addition; the addition operator must be commutative; there must be additive and multiplicative identities $\bar{0}$ and $\bar{1}$, such that $a \oplus \bar{0} = a$ and $a \otimes \bar{1} = a$; and there must be a multiplicative annihilator $\bar{0}$, such that $a \otimes \bar{0} = \bar{0}$.

Viterbi recurrence as,

$$v_m(k) = \bigoplus_{k' \in \mathcal{Y}} s_m(k, k') \otimes v_{m-1}(k'). \quad [7.109]$$

3850 Each recurrence that we have seen so far is a special case of this generalized Viterbi
 3851 recurrence:

- 3852 • In the max-product Viterbi recurrence over probabilities, the \oplus operation corresponds
 3853 to maximization, and the \otimes operation corresponds to multiplication.
- 3854 • In the forward recurrence over probabilities, the \oplus operation corresponds to addition,
 3855 and the \otimes operation corresponds to multiplication.
- 3856 • In the max-product Viterbi recurrence over log-probabilities, the \oplus operation corresponds
 3857 to maximization, and the \otimes operation corresponds to addition.¹²
- 3858 • In the forward recurrence over log-probabilities, the \oplus operation corresponds to
 3859 log-addition, $a \oplus b = \log(e^a + e^b)$. The \otimes operation corresponds to addition.

3860 The mathematical abstraction offered by semiring notation can be applied to the
 3861 software implementations of these algorithms, yielding concise and modular implementations.
 3862 For example, in the OpenFST library, generic operations are parametrized by the choice
 3863 of semiring (Allauzen et al., 2007).

3864 Exercises

- 3865 1. Extend the example in § 7.3.1 to the sentence they can can fish, meaning that “they
 3866 can put fish into cans.” Build the trellis for this example using the weights in Table 7.1,
 3867 and identify the best-scoring tag sequence. If the scores for noun and verb are tied,
 3868 then you may assume that the backpointer always goes to noun.
- 3869 2. Using the tagset $\mathcal{Y} = \{N, V\}$, and the feature set $f(\mathbf{w}, y_m, y_{m-1}, m) = \{(w_m, y_m), (y_m, y_{m-1})\}$,
 3870 show that there is no set of weights such that the correct tagging is obtained for both
 3871 they can fish (N V V) and they can can fish (N V V N).
- 3872 3. Work out what happens if you train a structured perceptron on the two examples
 3873 mentioned in the previous problem, using the transition and emission features (y_m, y_{m-1})
 3874 and (y_m, w_m) . Initialize all weights at 0, and assume that the Viterbi algorithm always
 3875 chooses N when the scores for the two tags are tied, so that the initial prediction for
 3876 they can fish is N N N.

¹²This is sometimes called the tropical semiring, in honor of the Brazilian mathematician Imre Simon.

- 3877 4. Consider the garden path sentence, The old man the boat. Given word-tag and
 3878 tag-tag features, what inequality in the weights must hold for the correct tag sequence
 3879 to outscore the garden path tag sequence for this example?
- 3880 5. Using the weights in Table 7.1, explicitly compute the log-probabilities for all possible
 3881 taggings of the input fish can. Verify that the forward algorithm recovers the aggregate
 3882 log probability.
- 3883 6. Sketch out an algorithm for a variant of Viterbi that returns the top- n label sequences.
 3884 What is the time and space complexity of this algorithm?
- 3885 7. Show how to compute the marginal probability $\Pr(y_{m-2} = k, y_m = k' \mid \mathbf{w}_{1:M})$, in
 3886 terms of the forward and backward variables, and the potentials $s_n(y_n, y_{n-1})$.
- 3887 8. Suppose you receive a stream of text, where some of tokens have been replaced at
 3888 random with NOISE. For example:
- 3889 • Source: I try all things, I achieve what I can
 - 3890 • Message received: I try NOISE NOISE, I NOISE what I NOISE
- 3891 Assume you have access to a pre-trained bigram language model, which gives probabilities
 3892 $p(w_m \mid w_{m-1})$. These probabilities can be assumed to be non-zero for all bigrams.
- 3893 a) Show how to use the Viterbi algorithm to try to recover the source by maximizing
 3894 the bigram language model log-probability. Specifically, set the scores $s_m(y_m, y_{m-1})$
 3895 so that the Viterbi algorithm selects a sequence of words that maximizes the
 3896 bigram language model log-probability, while leaving the non-noise tokens intact.
 3897 Your solution should not modify the logic of the Viterbi algorithm, it should
 3898 only set the scores $s_m(y_m, y_{m-1})$.
- 3899 b) An alternative solution is to iterate through the text from $m \in \{1, 2, \dots, M\}$,
 3900 replacing each noise token with the word that maximizes $P(w_m \mid w_{m-1})$ according
 3901 to the bigram language model. Given an upper bound on the expected fraction
 3902 of tokens for which the two approaches will disagree.
- 3903 9. Let $\alpha(\cdot)$ and $\beta(\cdot)$ indicate the forward and backward variables as defined in § 7.5.3.3.
 3904 Prove that $\alpha_{M+1}(\blacklozenge) = \beta_0(\lozenge) = \sum_y \alpha_m(y) \beta_m(y), \forall m \in \{1, 2, \dots, M\}$.
- 3905 10. Consider an RNN tagging model with a tanh activation function on the hidden layer,
 3906 and a hinge loss on the output. (The problem also works for the margin loss and
 3907 negative log-likelihood.) Suppose you initialize all parameters to zero: this includes
 3908 the word embeddings that make up \mathbf{x} , the transition matrix Θ , the output weights
 3909 β , and the initial hidden state \mathbf{h}_0 .

- 3910 a) Prove that for any data and for any gradient-based learning algorithm, all
3911 parameters will be stuck at zero.
- 3912 b) Would a sigmoid activation function avoid this problem?

3913 Chapter 8

3914 Applications of sequence labeling

3915 Sequence labeling has applications throughout natural language processing. This chapter
3916 focuses on part-of-speech tagging, morpho-syntactic attribute tagging, named entity recognition,
3917 and tokenization. It also touches briefly on two applications to interactive settings: dialogue
3918 act recognition and the detection of code-switching points between languages.

3919 8.1 Part-of-speech tagging

3920 The syntax of a language is the set of principles under which sequences of words are
3921 judged to be grammatically acceptable by fluent speakers. One of the most basic syntactic
3922 concepts is the part-of-speech (POS), which refers to the syntactic role of each word in a
3923 sentence. This concept was used informally in the previous chapter, and you may have
3924 some intuitions from your own study of English. For example, in the sentence We like
3925 vegetarian sandwiches, you may already know that we and sandwiches are nouns, like is
3926 a verb, and vegetarian is an adjective. These labels depend on the context in which the
3927 word appears: in she eats like a vegetarian, the word like is a preposition, and the word
3928 vegetarian is a noun.

3929 Parts-of-speech can help to disentangle or explain various linguistic problems. Recall
3930 Chomsky's proposed distinction in chapter 6:

3931 (8.1) Colorless green ideas sleep furiously.

3932 (8.2) *Ideas colorless furiously green sleep.

3933 One difference between these two examples is that the first contains part-of-speech transitions
3934 that are typical in English: adjective to adjective, adjective to noun, noun to verb, and verb
3935 to adverb. The second example contains transitions that are unusual: noun to adjective
3936 and adjective to verb. The ambiguity in a headline like,

3937 (8.3) Teacher Strikes Idle Children

3938 can also be explained in terms of parts of speech: in the interpretation that was likely
 3939 intended, strikes is a noun and idle is a verb; in the alternative explanation, strikes is a
 3940 verb and idle is an adjective.

3941 Part-of-speech tagging is often taken as a early step in a natural language processing
 3942 pipeline. Indeed, parts-of-speech provide features that can be useful for many of the
 3943 tasks that we will encounter later, such as parsing (chapter 10), coreference resolution
 3944 (chapter 15), and relation extraction (chapter 17).

3945 8.1.1 Parts-of-Speech

3946 The Universal Dependencies project (UD) is an effort to create syntactically-annotated
 3947 corpora across many languages, using a single annotation standard (Nivre et al., 2016). As
 3948 part of this effort, they have designed a part-of-speech tagset, which is meant to capture
 3949 word classes across as many languages as possible.¹ This section describes that inventory,
 3950 giving rough definitions for each of tags, along with supporting examples.

3951 Part-of-speech tags are morphosyntactic, rather than semantic, categories. This means
 3952 that they describe words in terms of how they pattern together and how they are internally
 3953 constructed (e.g., what suffixes and prefixes they include). For example, you may think of
 3954 a noun as referring to objects or concepts, and verbs as referring to actions or events. But
 3955 events can also be nouns:

3956 (8.4) ...the howling of the shrieking storm.

3957 Here howling and shrieking are events, but grammatically they act as a noun and adjective
 3958 respectively.

3959 8.1.1.1 The Universal Dependency part-of-speech tagset

3960 The UD tagset is broken up into three groups: open class tags, closed class tags, and
 3961 “others.”

3962 Open class tags Nearly all languages contain nouns, verbs, adjectives, and adverbs.²
 3963 These are all open word classes, because new words can easily be added to them. The
 3964 UD tagset includes two other tags that are open classes: proper nouns and interjections.

- 3965 • 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).

3966 (8.5) Toes are scarce among veteran blubber men.

3967 In English, nouns tend to follow determiners and adjectives, and can play the subject
3968 role in the sentence. They can be marked for the plural number by an -s suffix.

- 3969 • Proper nouns (PROPN) are tokens in names, which uniquely specify a given entity,

3970 (8.6) “Moby Dick?” shouted Ahab.

3971 • Verbs (VERB), according to the UD guidelines, “typically signal events and actions.”
3972 But they are also defined grammatically: they “can constitute a minimal predicate
3973 in a clause, and govern the number and types of other constituents which may occur
3974 in a clause.”³

3975 (8.7) “Moby Dick?” shouted Ahab.

3976 (8.8) Shall we keep chasing this murderous fish?

3977 English verbs tend to come in between the subject and some number of direct objects,
3978 depending on the verb. They can be marked for tense and aspect using suffixes such
3979 as -ed and -ing. (These suffixes are an example of inflectional morphology, which is
3980 discussed in more detail in § 9.1.4.)

- 3981 • Adjectives (ADJ) describe properties of entities,

3982 (8.9) Shall we keep chasing this murderous fish?

3983 (8.10) Toes are scarce among veteran blubber men.

3984 In the second example, scarce is a predicative adjective, linked to the subject by
3985 the copula verb are. In contrast, murderous and veteran are attributive adjectives,
3986 modifying the noun phrase in which they are embedded.

- 3987 • Adverbs (ADV) describe properties of events, and may also modify adjectives or other
3988 adverbs:

3989 (8.11) It is not down on any map; true places never are.

3990 (8.12) ...treacherously hidden beneath the loveliest tints of azure

3991 (8.13) Not drowned entirely, though.

- 3992 • Interjections (INTJ) are used in exclamations, e.g.,

3993 (8.14) Aye aye! it was that accursed white whale that razed me.

³<http://universaldependencies.org/u/pos/VERB.html>

3994 Closed class tags Closed word classes rarely receive new members. They are sometimes
 3995 referred to as function words — as opposed to content words — as they have little lexical
 3996 meaning of their own, but rather, help to organize the components of the sentence.

- 3997 • Adpositions (ADP) describe the relationship between a complement (usually a noun
 3998 phrase) and another unit in the sentence, typically a noun or verb phrase.

3999 (8.15) Toes are scarce among veteran blubber men.

4000 (8.16) It is not down on any map.

4001 (8.17) Give not thyself up then.

4002 As the examples show, English generally uses prepositions, which are adpositions
 4003 that appear before their complement. (An exception is ago, as in, we met three days
 4004 ago). Postpositions are used in other languages, such as Japanese and Turkish.

- 4005 • Auxiliary verbs (AUX) are a closed class of verbs that add information such as tense,
 4006 aspect, person, and number.

4007 (8.18) Shall we keep chasing this murderous fish?

4008 (8.19) What the white whale was to Ahab, has been hinted.

4009 (8.20) Ahab must use tools.

4010 (8.21) Meditation and water are wedded forever.

4011 (8.22) Toes are scarce among veteran blubber men.

4012 The final example is a copula verb, which is also tagged as an auxiliary in the UD
 4013 corpus.

- 4014 • Coordinating conjunctions (CCONJ) express relationships between two words or
 4015 phrases, which play a parallel role:

4016 (8.23) Meditation and water are wedded forever.

- 4017 • Subordinating conjunctions (SCONJ) link two clauses, making one syntactically
 4018 subordinate to the other:

4019 (8.24) It is the easiest thing in the world for a man to look as if he had a great
 4020 secret in him.

4021 Note that

- 4022 • Pronouns (PRON) are words that substitute for nouns or noun phrases.

4023 (8.25) Be it what it will, I'll go to it laughing.

4024 (8.26) I try all things, I achieve what I can.

4025 The example includes the personal pronouns I and it, as well as the relative pronoun
4026 what. Other pronouns include myself, somebody, and nothing.

- 4027 • Determiners (DET) provide additional information about the nouns or noun phrases
4028 that they modify:

4029 (8.27) What the white whale was to Ahab, has been hinted.

4030 (8.28) It is not down on any map.

4031 (8.29) I try all things ...

4032 (8.30) Shall we keep chasing this murderous fish?

4033 Determiners include articles (the), possessive determiners (their), demonstratives
4034 (this murderous fish), and quantifiers (any map).

- 4035 • Numerals (NUM) are an infinite but closed class, which includes integers, fractions,
4036 and decimals, regardless of whether spelled out or written in numerical form.

4037 (8.31) How then can this one small heart beat.

4038 (8.32) I am going to put him down for the three hundredth.

- 4039 • Particles (PART) are a catch-all of function words that combine with other words or
4040 phrases, but do not meet the conditions of the other tags. In English, this includes
4041 the infinitival to, the possessive marker, and negation.

4042 (8.33) Better to sleep with a sober cannibal than a drunk Christian.

4043 (8.34) So man's insanity is heaven's sense

4044 (8.35) It is not down on any map

4045 As the second example shows, the possessive marker is not considered part of the
4046 same token as the word that it modifies, so that man's is split into two tokens.
4047 (Tokenization is described in more detail in § 8.4.) A non-English example of a
4048 particle is the Japanese question marker ka:⁴

4049 (8.36) Sensei desu ka

Teacher is ?

4050 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.

4051 Other The remaining UD tags include punctuation (PUN) and symbols (SYM). Punctuation
 4052 is purely structural — e.g., commas, periods, colons — while symbols can carry content
 4053 of their own. Examples of symbols include dollar and percentage symbols, mathematical
 4054 operators, emoticons, emojis, and internet addresses. A final catch-all tag is X, which is
 4055 used for words that cannot be assigned another part-of-speech category. The X tag is also
 4056 used in cases of code switching (between languages), described in § 8.5.

4057 8.1.1.2 Other tagsets

4058 Prior to the Universal Dependency treebank, part-of-speech tagging was performed using
 4059 language-specific tagsets. The dominant tagset for English was designed as part of the Penn
 4060 Treebank (PTB), and it includes 45 tags — more than three times as many as the UD
 4061 tagset. This granularity is reflected in distinctions between singular and plural nouns, verb
 4062 tenses and aspects, possessive and non-possessive pronouns, comparative and superlative
 4063 adjectives and adverbs (e.g., faster, fastest), and so on. The Brown corpus includes a tagset
 4064 that is even more detailed, with 87 tags Francis (1964), including special tags for individual
 4065 auxiliary verbs such as be, do, and have.

4066 Different languages make different distinctions, and so the PTB and Brown tagsets are
 4067 not appropriate for a language such as Chinese, which does not mark the verb tense (Xia,
 4068 2000); nor for Spanish, which marks every combination of person and number in the verb
 4069 ending; nor for German, which marks the case of each noun phrase. Each of these languages
 4070 requires more detail than English in some areas of the tagset, and less in other areas. The
 4071 strategy of the Universal Dependencies corpus is to design a coarse-grained tagset to be used
 4072 across all languages, and then to additionally annotate language-specific morphosyntactic
 4073 attributes, such as number, tense, and case. The attribute tagging task is described in
 4074 more detail in § 8.2.

4075 Social media such as Twitter have been shown to require tagsets of their own (Gimpel
 4076 et al., 2011). Such corpora contain some tokens that are not equivalent to anything
 4077 encountered in a typical written corpus: e.g., emoticons, URLs, and hashtags. Social
 4078 media also includes dialectal words like gonna ('going to', e.g. We gonna be fine) and
 4079 Ima ('I'm going to', e.g., Ima tell you one more time), which can be analyzed either as
 4080 non-standard orthography (making tokenization impossible), or as lexical items in their
 4081 own right. In either case, it is clear that existing tags like noun and verb cannot handle
 4082 cases like Ima, which combine aspects of the noun and verb. Gimpel et al. (2011) therefore
 4083 propose a new set of tags to deal with these cases.

4084 8.1.2 Accurate part-of-speech tagging

4085 Part-of-speech tagging is the problem of selecting the correct tag for each word in a sentence.
 4086 Success is typically measured by accuracy on an annotated test set, which is simply the
 4087 fraction of tokens that were tagged correctly.

4088 8.1.2.1 Baselines

4089 A simple baseline for part-of-speech tagging is to choose the most common tag for each
4090 word. For example, in the Universal Dependencies treebank, the word talk appears 96
4091 times, and 85 of those times it is labeled as a verb: therefore, this baseline will always
4092 predict verb for this word. For words that do not appear in the training corpus, the baseline
4093 simply guesses the most common tag overall, which is noun. In the Penn Treebank, this
4094 simple baseline obtains accuracy above 92%. A more rigorous evaluation is the accuracy
4095 on out-of-vocabulary words, which are not seen in the training data. Tagging these words
4096 correctly requires attention to the context and the word’s internal structure.

4097 8.1.2.2 Contemporary approaches

4098 Conditional random fields and structured perceptron perform at or near the state-of-the-art
4099 for part-of-speech tagging in English. For example, (Collins, 2002) achieved 97.1% accuracy
4100 on the Penn Treebank, using a structured perceptron with the following base features
4101 (originally introduced by Ratnaparkhi (1996)):

- 4102 • current word, w_m
- 4103 • previous words, w_{m-1}, w_{m-2}
- 4104 • next words, w_{m+1}, w_{m+2}
- 4105 • previous tag, y_{m-1}
- 4106 • previous two tags, (y_{m-1}, y_{m-2})
- 4107 • for rare words:
 - 4108 – first k characters, up to $k = 4$
 - 4109 – last k characters, up to $k = 4$
 - 4110 – whether w_m contains a number, uppercase character, or hyphen.

4111 Similar results for the PTB data have been achieved using conditional random fields (CRFs;
4112 Toutanova et al., 2003).

4113 More recent work has demonstrated the power of neural sequence models, such as
4114 the long short-term memory (LSTM) (§ 7.6). Plank et al. (2016) apply a CRF and
4115 a bidirectional LSTM to twenty-two languages in the UD corpus, achieving an average
4116 accuracy of 94.3% for the CRF, and 96.5% with the bi-LSTM. Their neural model employs
4117 three types of embeddings: fine-tuned word embeddings, which are updated during training;
4118 pre-trained word embeddings, which are never updated, but which help to tag out-of-vocabulary
4119 words; and character-based embeddings. The character-based embeddings are computed
4120 by running an LSTM on the individual characters in each word, thereby capturing common
4121 orthographic patterns such as prefixes, suffixes, and capitalization. Extensive evaluations
4122 show that these additional embeddings are crucial to their model’s success.

word	PTB tag	UD tag	UD attributes
The	DT	DET	Definite=Def PronType=Art
German	JJ	ADJ	Degree=Pos
Expressionist	NN	NOUN	Number=Sing
movement	NN	NOUN	Number=Sing
was	VBD	AUX	Mood=Ind Number=Sing Person=3 Tense=Past VerbForm=Fin
destroyed	VBN	VERB	Tense=Past VerbForm=Part Voice=Pass
as	IN	ADP	
a	DT	DET	Definite=Ind PronType=Art
result	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.

4123 8.2 Morphosyntactic Attributes

4124 There is considerably more to say about a word than whether it is a noun or a verb:
 4125 in English, verbs are distinguish by features such tense and aspect, nouns by number,
 4126 adjectives by degree, and so on. These features are language-specific: other languages
 4127 distinguish other features, such as case (the role of the noun with respect to the action of
 4128 the sentence, which is marked in languages such as Latin and German⁵) and evidentiality
 4129 (the source of information for the speaker’s statement, which is marked in languages such
 4130 as Turkish). In the UD corpora, these attributes are annotated as feature-value pairs for
 4131 each token.⁶

4132 An example is shown in Figure 8.1. The determiner *the* is marked with two attributes:
 4133 *PronType=Art*, which indicates that it is an article (as opposed to another type of determiner

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).

4134 or pronominal modifier), and Definite=Def, which indicates that it is a definite article
4135 (referring to a specific, known entity). The verbs are each marked with several attributes.
4136 The auxiliary verb was is third-person, singular, past tense, finite (conjugated), and indicative
4137 (describing an event that has happened or is currently happenings); the main verb destroyed
4138 is in participle form (so there is no additional person and number information), past tense,
4139 and passive voice. Some, but not all, of these distinctions are reflected in the PTB tags
4140 VBD (past-tense verb) and VBN (past participle).

4141 While there are thousands of papers on part-of-speech tagging, there is comparatively
4142 little work on automatically labeling morphosyntactic attributes. Faruqui et al. (2016)
4143 train a support vector machine classification model, using a minimal feature set that
4144 includes the word itself, its prefixes and suffixes, and type-level information listing all
4145 possible morphosyntactic attributes for each word and its neighbors. Mueller et al. (2013)
4146 use a conditional random field (CRF), in which the tag space consists of all observed
4147 combinations of morphosyntactic attributes (e.g., the tag would be DEF+ART for the
4148 word the in Figure 8.1). This massive tag space is managed by decomposing the feature
4149 space over individual attributes, and pruning paths through the trellis. More recent work
4150 has employed bidirectional LSTM sequence models. For example, Pinter et al. (2017) train
4151 a bidirectional LSTM sequence model. The input layer and hidden vectors in the LSTM
4152 are shared across attributes, but each attribute has its own output layer, culminating
4153 in a softmax over all attribute values, e.g. $y_t^{\text{number}} \in \{\text{Sing}, \text{Plural}, \dots\}$. They find that
4154 character-level information is crucial, especially when the amount of labeled data is limited.

4155 Evaluation is performed by first computing recall and precision for each attribute. These
4156 scores can then be averaged at either the type or token level to obtain micro- or macro- F -
4157 measure. Pinter et al. (2017) evaluate on 23 languages in the UD treebank, reporting a
4158 median micro- F -measure of 0.95. Performance is strongly correlated with the size of the
4159 labeled dataset for each language, with a few outliers: for example, Chinese is particularly
4160 difficult, because although the dataset is relatively large (10^5 tokens in the UD 1.4 corpus),
4161 only 6% of tokens have any attributes, offering few useful labeled instances.

4162 8.3 Named Entity Recognition

4163 A classical problem in information extraction is to recognize and extract mentions of named
4164 entities in text. In news documents, the core entity types are people, locations, and
4165 organizations; more recently, the task has been extended to include amounts of money,
4166 percentages, dates, and times. In item 8.37 (Figure 8.2), the named entities include: The
4167 U.S. Army, an organization; Atlanta, a location; and May 14, 1864, a date. Named entity
4168 recognition is also a key task in biomedical natural language processing, with entity types
4169 including proteins, DNA, RNA, and cell lines (e.g., Collier et al., 2000; Ohta et al., 2002).
4170 Figure 8.2 shows an example from the GENIA corpus of biomedical research abstracts.

- (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).

4171 A standard approach to tagging named entity spans is to use discriminative sequence
 4172 labeling methods such as conditional random fields. However, the named entity recognition
 4173 (NER) task would seem to be fundamentally different from sequence labeling tasks like
 4174 part-of-speech tagging: rather than tagging each token, the goal is to recover spans of
 4175 tokens, such as The United States Army.

4176 This is accomplished by the BIO notation, shown in Figure 8.2. Each token at the
 4177 beginning of a name span is labeled with a B- prefix; each token within a name span
 4178 is labeled with an I- prefix. These prefixes are followed by a tag for the entity type, e.g.
 4179 B-Loc for the beginning of a location, and I-Protein for the inside of a protein name. Tokens
 4180 that are not parts of name spans are labeled as O. From this representation, the entity
 4181 name spans can be recovered unambiguously. This tagging scheme is also advantageous for
 4182 learning: tokens at the beginning of name spans may have different properties than tokens
 4183 within the name, and the learner can exploit this. This insight can be taken even further,
 4184 with special labels for the last tokens of a name span, and for unique tokens in name spans,
 4185 such as Atlanta in the example in Figure 8.2. This is called BILOU notation, and it can
 4186 yield improvements in supervised named entity recognition (Ratinov and Roth, 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)

4187 Another source of features is to use gazetteers: lists of known entity names. For example,
 4188 the U.S. Social Security Administration provides a list of tens of thousands of given names
 4189 — more than could be observed in any annotated corpus. Tokens or spans that match

- (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)

4190 an entry in a gazetteer can receive special features; this provides a way to incorporate
 4191 hand-crafted resources such as name lists in a learning-driven framework.

4192 Neural sequence labeling for NER Current research has emphasized neural sequence
 4193 labeling, using similar LSTM models to those employed in part-of-speech tagging (Hammerton,
 4194 2003; Huang et al., 2015; Lample et al., 2016). The bidirectional LSTM-CRF (Figure 7.4 in
 4195 § 7.6) does particularly well on this task, due to its ability to model tag-to-tag dependencies.
 4196 However, Strubell et al. (2017) show that convolutional neural networks can be equally
 4197 accurate, with significant improvement in speed due to the efficiency of implementing
 4198 ConvNets on graphics processing units (GPUs). The key innovation in this work was the
 4199 use of dilated convolution, which is described in more detail in § 3.4.

4200 8.4 Tokenization

4201 A basic problem for text analysis, first discussed in § 4.3.1, is to break the text into
 4202 a sequence of discrete tokens. For alphabetic languages such as English, deterministic
 4203 scripts suffice to achieve accurate tokenization. However, in logographic writing systems
 4204 such as Chinese script, words are typically composed of a small number of characters,
 4205 without intervening whitespace. The tokenization must be determined by the reader, with
 4206 the potential for occasional ambiguity, as shown in Figure 8.3. One approach is to match
 4207 character sequences against a known dictionary (e.g., Sproat et al., 1996), using additional
 4208 statistical information about word frequency. However, no dictionary is completely comprehensive,
 4209 and dictionary-based approaches can struggle with such out-of-vocabulary words.

4210 Chinese tokenization has therefore been approached as a supervised sequence labeling
 4211 problem. Xue et al. (2003) train a logistic regression classifier to make independent
 4212 segmentation decisions while moving a sliding window across the document. A set of rules
 4213 is then used to convert these individual classification decisions into an overall tokenization
 4214 of the input. However, these individual decisions may be globally suboptimal, motivating a
 4215 structure prediction approach. Peng et al. (2004) train a conditional random field to predict
 4216 labels of Start or NonStart on each character. More recent work has employed neural
 4217 network architectures. For example, Chen et al. (2015) use an LSTM-CRF architecture,

as described in § 7.6: they construct a trellis, in which each tag is scored according to the hidden state of an LSTM, and tag-tag transitions are scored according to learned transition weights. The best-scoring segmentation is then computed by the Viterbi algorithm.

4221 8.5 Code switching

4222 Multilingual speakers and writers do not restrict themselves to a single language. Code
4223 switching is the phenomenon of switching between languages in speech and text (Auer,
4224 2013; Poplack, 1980). Written code switching has become more common in online social
4225 media, as in the following extract from the website of Canadian President Justin Trudeau:⁷

4226 (8.39) Although everything written on this site est disponible en anglais
is available in English
4227 and in French, my personal videos seront bilingues
will be bilingual

4228 Accurately analyzing such texts requires first determining which languages are being used.
4229 Furthermore, quantitative analysis of code switching can provide insights on the languages
4230 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.

4239 Code switching is a special case of the more general problem of word level language
4240 identification, which Barman et al. (2014) address in the context of trilingual code switching
4241 between Bengali, English, and Hindi. They further observe an even more challenging
4242 phenomenon: intra-word code switching, such as the use of English suffixes with Bengali
4243 roots. They therefore mark each token as either (1) belonging to one of the three languages;
4244 (2) a mix of multiple languages; (3) “universal” (e.g., symbols, numbers, emoticons); or (4)
4245 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	So do you go college right now?
A	Abandoned	Are yo-
B	Yes-Answer	Yeah,
B	Statement	It's my last year [laughter].
A	Declarative-Question	You're a, so you're a senior now.
B	Yes-Answer	Yeah,
B	Statement	I'm working on my projects trying to graduate [laughter]
A	Appreciation	Oh, good for you.
B	Backchannel	Yeah.

Figure 8.4: An example of dialogue act labeling (Stolcke et al., 2000)

4246 8.6 Dialogue acts

4247 The sequence labeling problems that we have discussed so far have been over sequences of
 4248 word tokens or characters (in the case of tokenization). However, sequence labeling can
 4249 also be performed over higher-level units, such as utterances. Dialogue acts are labels over
 4250 utterances in a dialogue, corresponding roughly to the speaker’s intention — the utterance’s
 4251 illocutionary force (Austin, 1962). For example, an utterance may state a proposition (it
 4252 is not down on any map), pose a question (shall we keep chasing this murderous fish?),
 4253 or provide a response (aye aye!). Stolcke et al. (2000) describe how a set of 42 dialogue
 4254 acts were annotated for the 1,155 conversations in the Switchboard corpus (Godfrey et al.,
 4255 1992).⁸

4256 An example is shown in Figure 8.4. The annotation is performed over utterances, with
 4257 the possibility of multiple utterances per conversational turn (in cases such as interruptions,
 4258 an utterance may split over multiple turns). Some utterances are clauses (e.g., So do you
 4259 go to college right now?), while others are single words (e.g., yeah). Stolcke et al. (2000)
 4260 report that hidden Markov models (HMMs) achieve 96% accuracy on supervised utterance
 4261 segmentation. The labels themselves reflect the conversational goals of the speaker: the
 4262 utterance yeah functions as an answer in response to the question you’re a senior now, but
 4263 in the final line of the excerpt, it is a backchannel (demonstrating comprehension).

4264 For task of dialogue act labeling, Stolcke et al. (2000) apply a hidden Markov model.
 4265 The probability $p(\mathbf{w}_m | \mathbf{y}_m)$ must generate the entire sequence of words in the utterance,
 4266 and it is modeled as a trigram language model (§ 6.1). Stolcke et al. (2000) also account
 4267 for acoustic features, which capture the prosody of each utterance — for example, tonal
 4268 and rhythmic properties of speech, which can be used to distinguish dialogue acts such as

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.

4269 questions and answers. These features are handled with an additional emission distribution,
 4270 $p(\mathbf{a}_m \mid y_m)$, which is modeled with a probabilistic decision tree (Murphy, 2012). While
 4271 acoustic features yield small improvements overall, they play an important role in distinguishing
 4272 questions from statements, and agreements from backchannels.

4273 Recurrent neural architectures for dialogue act labeling have been proposed by Kalchbrenner
 4274 and Blunsom (2013) and Ji et al. (2016), with strong empirical results. Both models are
 4275 recurrent at the utterance level, so that each complete utterance updates a hidden state.
 4276 The recurrent-convolutional network of Kalchbrenner and Blunsom (2013) uses convolution
 4277 to obtain a representation of each individual utterance, while Ji et al. (2016) use a second
 4278 level of recurrence, over individual words. This enables their method to also function as a
 4279 language model, giving probabilities over sequences of words in a document.

4280 Exercises

- 4281 1. Using the Universal Dependencies part-of-speech tags, annotate the following sentences.
 4282 You may examine the UD tagging guidelines. Tokenization is shown with whitespace.
 4283 Don't forget about punctuation.

4284 (8.40) I try all things , I achieve what I can .
 4285 (8.41) It was that accursed white whale that razed me .
 4286 (8.42) Better to sleep with a sober cannibal , than a drunk Christian .
 4287 (8.43) Be it what it will , I 'll go to it laughing .

- 4288 2. Select three short sentences from a recent news article, and annotate them for UD
 4289 part-of-speech tags. Ask a friend to annotate the same three sentences without
 4290 looking at your annotations. Compute the rate of agreement, using the Kappa metric
 4291 defined in § 4.5.2.1. Then work together to resolve any disagreements.
- 4292 3. Choose one of the following morphosyntactic attributes: Mood, Tense, Voice. Research
 4293 the definition of this attribute on the universal dependencies website, <http://universaldependencies.org/u/feat/index.html>. Returning to the examples in the first exercise, annotate all
 4294 verbs for your chosen attribute. It may be helpful to consult examples from an
 4295 English-language universal dependencies corpus, available at https://github.com/UniversalDependencies/UD_English-EWT/tree/master.
- 4296 4. Download a dataset annotated for universal dependencies, such as the English Treebank
 4297 at https://github.com/UniversalDependencies/UD_English-EWT/tree/master. This
 4298 corpus is already segmented into training, development, and test data.

- 4301 a) First, train a logistic regression or SVM classifier using character suffixes: character
 4302 n-grams up to length 4. Compute the recall, precision, and *F*-measure on the
 4303 development data.
- 4304 b) Next, augment your classifier using the same character suffixes of the preceding
 4305 and succeeding tokens. Again, evaluate your classifier on heldout data.
- 4306 c) Optionally, train a Viterbi-based sequence labeling model, using a toolkit such
 4307 as CRFSuite (<http://www.chokkan.org/software/crfsuite/>) or your own Viterbi
 4308 implementation. This is more likely to be helpful for attributes in which agreement
 4309 is required between adjacent words. For example, in Romance languages, determiners,
 4310 nouns, and adjectives must agree in gender and number.
- 4311 5. Provide BIO-style annotation of the named entities (person, place, organization, date,
 4312 or product) in the following expressions:
- 4313 (8.44) The third mate was Flask, a native of Tisbury, in Martha’s Vineyard.
 4314 (8.45) Its official Nintendo announced today that they Will release the Nintendo
 4315 3DS in north America march 27 (Ritter et al., 2011).
 4316 (8.46) Jessica Reif, a media analyst at Merrill Lynch & Co., said, “If they can get
 4317 up and running with exclusive programming within six months, it doesn’t
 4318 set the venture back that far.”⁹
- 4319 6. Run the examples above through the online version of a named entity recognition
 4320 tagger, such as the Allen NLP system here: <http://demo.allennlp.org/named-entity-recognition>.
 4321 Do the predicted tags match your annotations?
- 4322 7. Build a whitespace tokenizer for English:
- 4323 a) Using the NLTK library, download the complete text to the novel Alice in
 4324 Wonderland (Carroll, 1865). Hold out the final 1000 words as a test set.
- 4325 b) Label each alphanumeric character as a segmentation point, $y_m = 1$ if m is
 4326 the final character of a token. Label every other character as $y_m = 0$. Then
 4327 concatenate all the tokens in the training and test sets. Make sure that the
 4328 number of labels $\{y_m\}_{m=1}^M$ is identical to the number of characters $\{c_m\}_{m=1}^M$
 4329 in your concatenated datasets.
- 4330 c) Train a logistic regression classifier to predict y_m , using the surrounding characters
 4331 $c_{m-5:m+5}$ as features. After training the classifier, run it on the test set, using
 4332 the predicted segmentation points to re-tokenize the text.
- 4333 d) Compute the per-character segmentation accuracy on the test set. You should
 4334 be able to get at least 88% accuracy.

⁹From the Message Understanding Conference (MUC-7) dataset (Chinchor and Robinson, 1997).

4335 e) Print out a sample of segmented text from the test set, e.g.

4336 Thereareno mice in the air , I ' m afraid , but y oumight cat
4337 chabat , and that ' s very like a mouse , youknow . But
4338 docatseat bats , I wonder ?'

4339 8. Perform the following extensions to the previous problem:

- 4340 a) Train a conditional random field sequence labeler, by incorporating the tag
4341 bigrams (y_{m-1}, y_m) as additional features. You may use a structured prediction
4342 library such as CRFSuite (<http://www.chokkan.org/software/crfsuite/>), or you
4343 may want to implement Viterbi yourself. Compare the accuracy with your
4344 classification-based approach.
- 4345 b) Optionally, compute the token-level performance. Treating the original tokenization
4346 as ground truth, compute the number of true positives (tokens that are in both
4347 the ground truth and predicted tokenization), false positives (tokens that are in
4348 the predicted tokenization but not the ground truth), and false negatives (tokens
4349 that are in the ground truth but not the predicted tokenization). Compute the
4350 F-measure.
4351 Hint: to match predicted and ground truth tokens, add “anchors” for the start
4352 character of each token. The number of true positives is then the size of the
4353 intersection of the sets of predicted and ground truth tokens.
- 4354 c) Apply the same methodology in a more practical setting: tokenization of Chinese,
4355 which is written without whitespace. You can find annotated datasets at <http://alias-i.com/lingpipe/demos/tutorial/chineseTokens/read-me.html>.

4357 Chapter 9

4358 Formal language theory

4359 We have now seen methods for learning to label individual words, vectors of word counts,
4360 and sequences of words; we will soon proceed to more complex structural transformations.
4361 Most of these techniques could apply to counts or sequences from any discrete vocabulary;
4362 there is nothing fundamentally linguistic about, say, a hidden Markov model. This raises
4363 a basic question that this text has not yet considered: what is a language?

4364 This chapter will take the perspective of formal language theory, in which a language
4365 is defined as a set of strings, each of which is a sequence of elements from a finite alphabet.
4366 For interesting languages, there are an infinite number of strings that are in the language,
4367 and an infinite number of strings that are not. For example:

- 4368 • the set of all even-length sequences from the alphabet $\{a, b\}$, e.g., $\{\emptyset, aa, ab, ba, bb, aaaa, aaab, \dots\}$;
- 4369 • the set of all sequences from the alphabet $\{a, b\}$ that contain aaa as a substring, e.g.,
4370 $\{aaa, aaaa, baaa, aaab, \dots\}$;
- 4371 • the set of all sequences of English words (drawn from a finite dictionary) that contain
4372 at least one verb (a finite subset of the dictionary);
- 4373 • the python programming language.

4374 Formal language theory defines classes of languages and their computational properties.
4375 Of particular interest is the computational complexity of solving the membership problem
4376 — determining whether a string is in a language. The chapter will focus on three classes
4377 of formal languages: regular, context-free, and “mildly” context-sensitive languages.

4378 A key insight of 20th century linguistics is that formal language theory can be usefully
4379 applied to natural languages such as English, by designing formal languages that capture
4380 as many properties of the natural language as possible. For many such formalisms, a
4381 useful linguistic analysis comes as a byproduct of solving the membership problem. The

4382 membership problem can be generalized to the problems of scoring strings for their acceptability
 4383 (as in language modeling), and of transducing one string into another (as in translation).

4384 **9.1 Regular languages**

4385 Sooner or later, most computer scientists will write a regular expression. If you have, then
 4386 you have defined a regular language, which is any language that can be defined by a regular
 4387 expression. Formally, a regular expression can include the following elements:

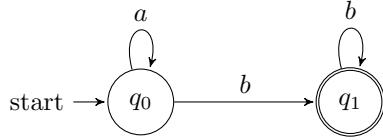
- 4388 • A literal character drawn from some finite alphabet Σ .
- 4389 • The empty string ϵ .
- 4390 • The concatenation of two regular expressions RS , where R and S are both regular
 4391 expressions. The resulting expression accepts any string that can be decomposed
 4392 $x = yz$, where y is accepted by R and z is accepted by S .
- 4393 • The alternation $R | S$, where R and S are both regular expressions. The resulting
 4394 expression accepts a string x if it is accepted by R or it is accepted by S .
- 4395 • The Kleene star R^* , which accepts any string x that can be decomposed into a
 4396 sequence of strings which are all accepted by R .
- 4397 • Parenthesization (R) , which is used to limit the scope of the concatenation, alternation,
 4398 and Kleene star operators.

4399 Here are some example regular expressions:

- 4400 • The set of all even length strings on the alphabet $\{a, b\}$: $((aa)|(ab)|(ba)|(bb))^*$
- 4401 • The set of all sequences of the alphabet $\{a, b\}$ that contain aaa as a substring:
 4402 $(a|b)^*aaa(a|b)^*$
- 4403 • The set of all sequences of English words that contain at least one verb: W^*VW^* ,
 4404 where W is an alternation between all words in the dictionary, and V is an alternation
 4405 between all verbs ($V \subseteq W$).

4406 This list does not include a regular expression for the Python programming language,
 4407 because this language is not regular — there is no regular expression that can capture its
 4408 syntax. We will discuss why towards the end of this section.

4409 Regular languages are closed under union, intersection, and concatenation. This means,
 4410 for example, that if two languages L_1 and L_2 are regular, then so are the languages $L_1 \cup L_2$,
 4411 $L_1 \cap L_2$, and the language of strings that can be decomposed as $s = tu$, with $s \in L_1$ and
 4412 $t \in L_2$. Regular languages are also closed under negation: if L is regular, then so is the
 4413 language $\bar{L} = \{s \notin L\}$.

Figure 9.1: State diagram for the finite state acceptor M_1 .

4414 9.1.1 Finite state acceptors

4415 A regular expression defines a regular language, but does not give an algorithm for determining
 4416 whether a string is in the language that it defines. Finite state automata are theoretical
 4417 models of computation on regular languages, which involve transitions between a finite
 4418 number of states. The most basic type of finite state automaton is the finite state acceptor
 4419 (FSA), which describes the computation involved in testing if a string is a member of a
 4420 language. Formally, a finite state acceptor is a tuple $M = (Q, \Sigma, q_0, F, \delta)$, consisting of:

- 4421 • a finite alphabet Σ of input symbols;
- 4422 • a finite set of states $Q = \{q_0, q_1, \dots, q_n\}$;
- 4423 • a start state $q_0 \in Q$;
- 4424 • a set of final states $F \subseteq Q$;
- 4425 • a transition function $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$. The transition function maps from a
 4426 state and an input symbol (or empty string ϵ) to a set of possible resulting states.

4427 A path in M is a sequence of transitions, $\pi = t_1, t_2, \dots, t_N$, where each t_i traverses an
 4428 arc in the transition function δ . The finite state acceptor M accepts a string ω if there is
 4429 an accepting path, in which the initial transition t_1 begins at the start state q_0 , the final
 4430 transition t_N terminates in a final state in Q , and the entire input ω is consumed.

4431 9.1.1.1 Example

Consider the following FSA, M_1 .

$$\Sigma = \{a, b\} \quad [9.1]$$

$$Q = \{q_0, q_1\} \quad [9.2]$$

$$F = \{q_1\} \quad [9.3]$$

$$\delta = \{(q_0, a) \rightarrow q_0, (q_0, b) \rightarrow q_1, (q_1, b) \rightarrow q_1\}. \quad [9.4]$$

4432 This FSA defines a language over an alphabet of two symbols, a and b. The transition
 4433 function δ is written as a set of arcs: $(q_0, a) \rightarrow q_0$ says that if the machine is in state
 4434 q_0 and reads symbol a, it stays in q_0 . Figure 9.1 provides a graphical representation of

4435 M_1 . Because each pair of initial state and symbol has at most one resulting state, M_1 is
 4436 deterministic: each string ω induces at most one accepting path. Note that there are no
 4437 transitions for the symbol a in state q_1 ; if a is encountered in q_1 , then the acceptor is stuck,
 4438 and the input string is rejected.

4439 What strings does M_1 accept? The start state is q_0 , and we have to get to q_1 , since this
 4440 is the only final state. Any number of a symbols can be consumed in q_0 , but a b symbol is
 4441 required to transition to q_1 . Once there, any number of b symbols can be consumed, but
 4442 an a symbol cannot. So the regular expression corresponding to the language defined by
 4443 M_1 is a^*bb^* .

4444 9.1.1.2 Computational properties of finite state acceptors

4445 The key computational question for finite state acceptors is: how fast can we determine
 4446 whether a string is accepted? For deterministic FSAs, this computation can be performed
 4447 by Dijkstra's algorithm, with time complexity $\mathcal{O}(V \log V + E)$, where V is the number of
 4448 vertices in the FSA, and E is the number of edges (Cormen et al., 2009). Non-deterministic
 4449 FSAs (NFSAs) can include multiple transitions from a given symbol and state. Any NSFA
 4450 can be converted into a deterministic FSA, but the resulting automaton may have a number
 4451 of states that is exponential in the number of size of the original NFSA (Mohri et al., 2002).

4452 9.1.2 Morphology as a regular language

4453 Many words have internal structure, such as prefixes and suffixes that shape their meaning.
 4454 The study of word-internal structure is the domain of morphology, of which there are two
 4455 main types:

- 4456 • Derivational morphology describes the use of affixes to convert a word from one
 4457 grammatical category to another (e.g., from the noun grace to the adjective graceful),
 4458 or to change the meaning of the word (e.g., from grace to disgrace).
- 4459 • Inflectional morphology describes the addition of details such as gender, number,
 4460 person, and tense (e.g., the -ed suffix for past tense in English).

4461 Morphology is a rich topic in linguistics, deserving of a course in its own right.¹ The
 4462 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 as 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).

4463 current section deals with derivational morphology; inflectional morphology is discussed in
4464 § 9.1.4.3.

4465 Suppose that we want to write a program that accepts only those words that are
4466 constructed in accordance with the rules of English derivational morphology:

- 4467 (9.1) grace, graceful, gracefully, *gracelyful
4468 (9.2) disgrace, *ungrace, disgraceful, disgracefully
4469 (9.3) allure, *allureful, alluring, alluringly
4470 (9.4) fairness, unfair, *disfair, fairly

4471 (Recall that the asterisk indicates that a linguistic example is judged unacceptable by fluent
4472 speakers of a language.) These examples cover only a tiny corner of English derivational
4473 morphology, but a number of things stand out. The suffix -ful converts the nouns grace and
4474 disgrace into adjectives, and the suffix -ly converts adjectives into adverbs. These suffixes
4475 must be applied in the correct order, as shown by the unacceptability of *gracelyful. The
4476 -ful suffix works for only some words, as shown by the use of alluring as the adjetival form
4477 of allure. Other changes are made with prefixes, such as the derivation of disgrace from
4478 grace, which roughly corresponds to a negation; however, fair is negated with the un- prefix
4479 instead. Finally, while the first three examples suggest that the direction of derivation is
4480 noun → adjective → adverb, the example of fair suggests that the adjective can also be
4481 the base form, with the -ness suffix performing the conversion to a noun.

4482 Can we build a computer program that accepts only well-formed English words, and
4483 rejects all others? This might at first seem trivial to solve with a brute-force attack:
4484 simply make a dictionary of all valid English words. But such an approach fails to account
4485 for morphological productivity — the applicability of existing morphological rules to new
4486 words and names, such as Trump to Trumpy and Trumpkin, and Clinton to Clintonian
4487 and Clintonite. We need an approach that represents morphological rules explicitly, and
4488 for this we will try a finite state acceptor.

4489 The dictionary approach can be implemented as a finite state acceptor, with the
4490 vocabulary Σ equal to the vocabulary of English, and a transition from the start state
4491 to the accepting state for each word. But this would of course fail to generalize beyond the
4492 original vocabulary, and would not capture anything about the morphotactic rules that
4493 govern derivations from new words. The first step towards a more general approach is
4494 shown in Figure 9.2, which is the state diagram for a finite state acceptor in which the
4495 vocabulary consists of morphemes, which include stems (e.g., grace, allure) and affixes
4496 (e.g., dis-, -ing, -ly). This finite state acceptor consists of a set of paths leading away from
4497 the start state, with derivational affixes added along the path. Except for q_{neg} , the states
4498 on these paths are all final, so the FSA will accept disgrace, disgraceful, and disgracefully,
4499 but not dis-.

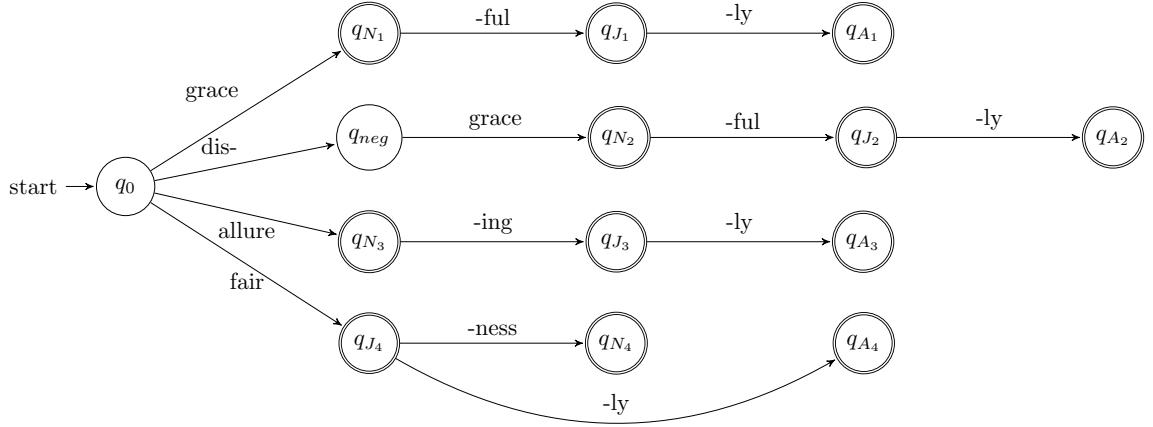


Figure 9.2: A finite state acceptor for a fragment of English derivational morphology. Each path represents possible derivations from a single root form.

4500 This FSA can be minimized to the form shown in Figure 9.3, which makes the generality
 4501 of the finite state approach more apparent. For example, the transition from q_0 to q_{J_2} can
 4502 be made to accept not only fair but any single-morpheme (monomorphemic) adjective
 4503 that takes -ness and -ly as suffixes. In this way, the finite state acceptor can easily be
 4504 extended: as new word stems are added to the vocabulary, their derived forms will be
 4505 accepted automatically. Of course, this FSA would still need to be extended considerably
 4506 to cover even this small fragment of English morphology. As shown by cases like music →
 4507 musical, athlete → athletic, English includes several classes of nouns, each with its own
 4508 rules for derivation.

4509 The FSAs shown in Figure 9.2 and 9.3 accept alluring, not alluring. This reflects a
 4510 distinction between morphology — the question of which morphemes to use, and in what
 4511 order — and orthography — the question of how the morphemes are rendered in written
 4512 language. Just as orthography requires dropping the e preceding the -ing suffix, phonology
 4513 imposes a related set of constraints on how words are rendered in speech. As we will
 4514 see soon, these issues are handled through finite state transducers, which are finite state
 4515 automata that take inputs and produce outputs.

4516 9.1.3 Weighted finite state acceptors

4517 According to the FSA treatment of morphology, every word is either in or out of the
 4518 language, with no wiggle room. Perhaps you agree that musicky and fishful are not valid
 4519 English words; but if forced to choose, you probably find a fishful stew or a musicky tribute
 4520 preferable to behaving disgracelyful. Rather than asking whether a word is acceptable, we
 4521 might like to ask how acceptable it is. Aronoff (1976, page 36) puts it another way: “Though

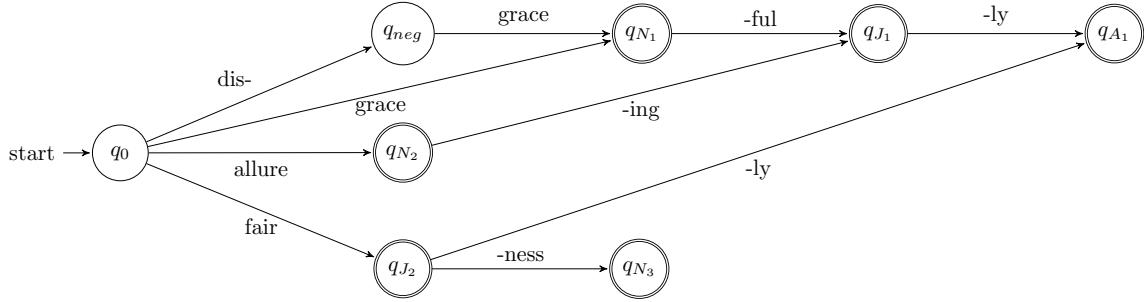


Figure 9.3: Minimization of the finite state acceptor shown in Figure 9.2.

many things are possible in morphology, some are more possible than others.” But finite state acceptors give no way to express preferences among technically valid choices.

Weighted finite state acceptors (WFSAs) are generalizations of FSAs, in which each accepting path is assigned a score, computed from the transitions, the initial state, and the final state. Formally, a weighted finite state acceptor $M = (Q, \Sigma, \lambda, \rho, \delta)$ consists of:

- a finite set of states $Q = \{q_0, q_1, \dots, q_n\}$;
- a finite alphabet Σ of input symbols;
- an initial weight function, $\lambda : Q \mapsto \mathbb{R}$;
- a final weight function $\rho : Q \mapsto \mathbb{R}$;
- a transition function $\delta : Q \times \Sigma \times Q \mapsto \mathbb{R}$.

WFSAs depart from the FSA formalism in three ways: every state can be an initial state, with score $\lambda(q)$; every state can be an accepting state, with score $\rho(q)$; transitions are possible between any pair of states on any input, with a score $\delta(q_i, \omega, q_j)$. Nonetheless, FSAs can be viewed as a special case: for any FSA M we can build an equivalent WPSA 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 transitions $\{(q_1, \omega) \rightarrow q_2\}$ that are not permitted by the transition function of M .

The total score for any path $\pi = t_1, t_2, \dots, t_N$ is equal to the sum of these scores,

$$d(\pi) = \lambda(\text{from-state}(t_1)) + \sum_n^N \delta(t_n) + \rho(\text{to-state}(t_N)). \quad [9.5]$$

A shortest-path algorithm is used to find the minimum-cost path through a WPSA for string ω , with time complexity $\mathcal{O}(E + V \log V)$, where E is the number of edges and V is 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

4542 9.1.3.1 N-gram language models as WFSAs

4543 In n -gram language models (see § 6.1), the probability of a sequence of tokens w_1, w_2, \dots, w_M
 4544 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:

$$\begin{aligned} \delta(q_i, \omega, q_j) &= \begin{cases} \log \Pr(w_m = j | w_{m-1} = i), & \omega = j \\ -\infty, & \omega \neq j \end{cases} \\ \lambda(q_i) &= \log \Pr(w_1 = i | w_0 = \square) \\ \rho(q_i) &= \log \Pr(w_{M+1} = \blacksquare | w_M = i). \end{aligned}$$

4545 The transition function is designed to ensure that the context is recorded accurately:
 4546 we can move to state j on input ω only if $\omega = j$; otherwise, transitioning to state j is
 4547 forbidden by the weight of $-\infty$. The initial weight function $\lambda(q_i)$ is the log probability of
 4548 receiving i as the first token, and the final weight function $\rho(q_i)$ is the log probability of
 4549 receiving an “end-of-string” token after observing $w_M = i$.

4550 9.1.3.2 *Semiring weighted finite state acceptors

4551 The n -gram language model WFSA is deterministic: each input has exactly one accepting
 4552 path, for which the WFSA computes a score. In non-deterministic WFSAs, a given

log 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.”

4553 input may have multiple accepting paths. In some applications, the score for the input is
 4554 aggregated across all such paths. Such aggregate scores can be computed by generalizing
 4555 WFSAs with semiring notation, first introduced in § 7.7.3.

4556 Let $d(\pi)$ represent the total score for path $\pi = t_1, t_2, \dots, t_N$, which is computed as,

$$d(\pi) = \lambda(\text{from-state}(t_1)) \otimes \delta(t_1) \otimes \delta(t_2) \otimes \dots \otimes \delta(t_N) \otimes \rho(\text{to-state}(t_N)). \quad [9.8]$$

4557 This is a generalization of Equation 9.5 to semiring notation, using the semiring multiplication
 4558 operator \otimes in place of addition.

4559 Now let $s(\omega)$ represent the total score for all paths $\Pi(\omega)$ that consume input ω ,

$$s(\omega) = \bigoplus_{\pi \in \Pi(\omega)} d(\pi). \quad [9.9]$$

4560 Here, semiring addition (\oplus) is used to combine the scores of multiple paths.

4561 The generalization to semirings covers a number of useful special cases. In the log-probability
 4562 semiring, multiplication is defined as $\log p(x) \otimes \log p(y) = \log p(x) + \log p(y)$, and addition
 4563 is defined as $\log p(x) \oplus \log p(y) = \log(p(x) + p(y))$. Thus, $s(\omega)$ represents the log-probability
 4564 of accepting input ω , marginalizing over all paths $\pi \in \Pi(\omega)$. In the boolean semiring, the
 4565 \otimes operator is logical conjunction, and the \oplus operator is logical disjunction. This reduces
 4566 to the special case of unweighted finite state acceptors, where the score $s(\omega)$ is a boolean
 4567 indicating whether there exists any accepting path for ω . In the tropical semiring, the \oplus
 4568 operator is a maximum, so the resulting score is the score of the best-scoring path through
 4569 the WFSAs. The OpenFST toolkit uses semirings and polymorphism to implement general
 4570 algorithms for weighted finite state automata (Allauzen et al., 2007).

4571 9.1.3.3 *Interpolated n -gram language models

4572 Recall from § 6.2.3 that an interpolated n -gram language model combines the probabilities
 4573 from multiple n -gram models. For example, an interpolated bigram language model
 4574 computes probability,

$$\hat{p}(w_m | w_{m-1}) = \lambda_1 p_1(w_m) + \lambda_2 p_2(w_m | w_{m-1}), \quad [9.10]$$

4575 with \hat{p} indicating the interpolated probability, p_2 indicating the bigram probability, and
 4576 p_1 indicating the unigram probability. We set $\lambda_2 = (1 - \lambda_1)$ so that the probabilities sum
 4577 to one.

4578 Interpolated bigram language models can be implemented using a non-deterministic
 4579 WFSAs (Knight and May, 2009). The basic idea is shown in Figure 9.4. In an interpolated
 4580 bigram language model, there is one state for each element in the vocabulary — in this
 4581 case, the states q_A and q_B — which capture the contextual conditioning in the bigram

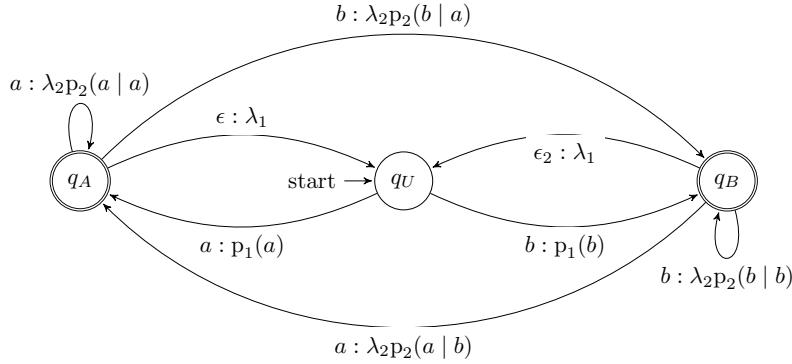


Figure 9.4: WFSA implementing an interpolated bigram/unigram language model, on the alphabet $\Sigma = \{a, b\}$. For simplicity, the WFSA is constrained to force the first token to be generated from the unigram model, and does not model the emission of the end-of-sequence token.

probabilities. To model unigram probabilities, there is an additional state q_U , which “forgets” the context. Transitions out of q_U involve unigram probabilities, $p_1(a)$ and $p_2(b)$; transitions into q_U emit the empty symbol ϵ , and have probability λ_1 , reflecting the interpolation weight for the unigram model. The interpolation weight for the bigram model is included in the weight of the transition $q_A \rightarrow q_B$.

The epsilon transitions into q_U make this WFSA non-deterministic. Consider the score for the sequence (a, b, b) . The initial state is q_U , so the symbol a is generated with score $p_1(a)$ ³ Next, we can generate b from the unigram model by taking the transition $q_A \rightarrow q_B$, with score $\lambda_2 p_2(b | a)$. Alternatively, we can take a transition back to q_U with score λ_1 , and then emit b from the unigram model with score $p_1(b)$. To generate the final b token, we face the same choice: emit it directly from the self-transition to q_B , or transition to q_U first.

The total score for the sequence (a, b, b) is the semiring sum over all accepting paths,

$$\begin{aligned}
 s(a, b, b) = & (p_1(a) \otimes \lambda_2 p_2(b | a) \otimes \lambda_2 p(b | b)) \\
 & \oplus (p_1(a) \otimes \lambda_1 \otimes p_1(b) \otimes \lambda_2 p(b | b)) \\
 & \oplus (p_1(a) \otimes \lambda_2 p_2(b | a) \otimes p_1(b) \otimes p_1(b)) \\
 & \oplus (p_1(a) \otimes \lambda_1 \otimes p_1(b) \otimes p_1(b) \otimes p_1(b)). \tag{[9.11]}
 \end{aligned}$$

Each line in Equation 9.11 represents the probability of a specific path through the WFSA. 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.

4596 transition weight, which are themselves probabilities. The \oplus operator is addition, so that
 4597 the total score is the sum of the scores (probabilities) for each path. This corresponds to
 4598 the probability under the interpolated bigram language model.

4599 9.1.4 Finite state transducers

4600 Finite state acceptors can determine whether a string is in a regular language, and weighted
 4601 finite state acceptors can compute a score for every string over a given alphabet. Finite
 4602 state transducers (FSTs) extend the formalism further, by adding an output symbol to
 4603 each transition. Formally, a finite state transducer is a tuple $T = (Q, \Sigma, \Omega, \lambda, \rho, \delta)$, with
 4604 Ω representing an output vocabulary and the transition function $\delta : Q \times (\Sigma \cup \epsilon) \times (\Omega \cup$
 4605 $\epsilon) \times Q \rightarrow \mathbb{R}$ mapping from states, input symbols, and output symbols to states. The
 4606 remaining elements $(Q, \Sigma, \lambda, \rho)$ are identical to their definition in weighted finite state
 4607 acceptors (§ 9.1.3). Thus, each path through the FST T transduces the input string into
 4608 an output.

4609 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,

$$\delta(q, a, a, q) = \delta(q, b, b, q) = 0 \quad [9.12]$$

$$\delta(q, a, b, q) = \delta(q, b, a, q) = 1 \quad [9.13]$$

$$\delta(q, a, \epsilon, q) = \delta(q, b, \epsilon, q) = 1 \quad [9.14]$$

$$\delta(q, \epsilon, a, q) = \delta(q, \epsilon, b, q) = 1. \quad [9.15]$$

4610 The state diagram is shown in Figure 9.5.

4611 For a given string pair, there are multiple paths through the transducer: the best-scoring
 4612 path from dessert to desert involves a single deletion, for a total score of 1; the worst-scoring
 4613 path involves seven deletions and six additions, for a score of 13.

4614 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

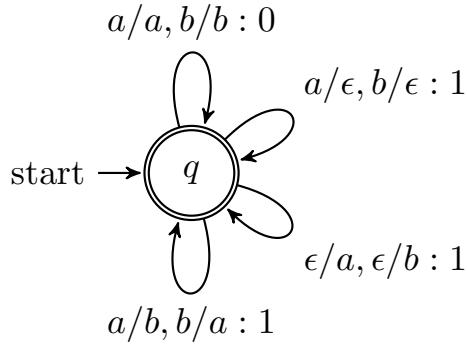


Figure 9.5: State diagram for the Levenshtein edit distance finite state transducer. The label $x/y : c$ indicates a cost of c for a transition with input x and output y .

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]

4615 The final two lines appear to conflict; they are meant to be interpreted as an instruction
 4616 to remove a terminal -s unless it is part of an -ss ending. A state diagram to handle just
 4617 these final two lines is shown in Figure 9.6. Make sure you understand how this finite state
 4618 transducer handles cats, steps, bass, and basses.

4619 9.1.4.3 Inflectional morphology

4620 In inflectional morphology, word lemmas are modified to add grammatical information such
 4621 as tense, number, and case. For example, many English nouns are pluralized by the suffix
 4622 -s, and many verbs are converted to past tense by the suffix -ed. English's inflectional
 4623 morphology is considerably simpler than many of the world's languages. For example,
 4624 Romance languages (derived from Latin) feature complex systems of verb suffixes which
 4625 must agree with the person and number of the verb, as shown in Table 9.1.

4626 The task of morphological analysis is to read a form like *canto*, and output an analysis
 4627 like *cantar+Verb+PresInd+1p+Sing*, where +PresInd describes the tense as present indicative,
 4628 +1p indicates the first-person, and +Sing indicates the singular number. The task of
 4629 morphological generation is the reverse, going from *cantar+Verb+PresInd+1p+Sing* to
 4630 *canto*. Finite state transducers are an attractive solution, because they can solve both
 4631 problems with a single model (Beesley and Karttunen, 2003). As an example, Figure 9.7
 4632 shows a fragment of a finite state transducer for Spanish inflectional morphology. The input

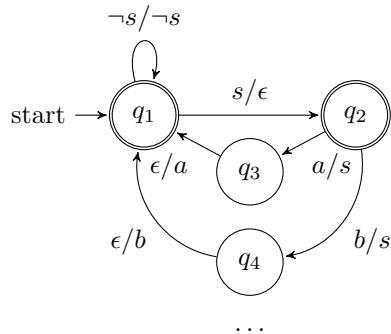


Figure 9.6: State diagram for final two lines of step 1a of the Porter stemming diagram. States q_3 and q_4 “remember” the observations a and b respectively; the ellipsis \dots represents additional states for each symbol in the input alphabet. The notation $\neg s / \neg s$ is not part of the FST formalism; 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)	antas	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.

4633 vocabulary Σ corresponds to the set of letters used in Spanish spelling, and the output
 4634 vocabulary Ω corresponds to these same letters, plus the vocabulary of morphological
 4635 features (e.g., +Sing, +Verb). In Figure 9.7, there are two paths that take *canto* as input,
 4636 corresponding to the verb and noun meanings; the choice between these paths could be
 4637 guided by a part-of-speech tagger. By inversion, the inputs and outputs for each transition
 4638 are switched, resulting in a finite state generator, capable of producing the correct surface
 4639 form for any morphological analysis.

4640 Finite state morphological analyzers and other unweighted transducers can be designed
 4641 by hand. The designer’s goal is to avoid overgeneration — accepting strings or making
 4642 transductions that are not valid in the language — as well as undergeneration — failing

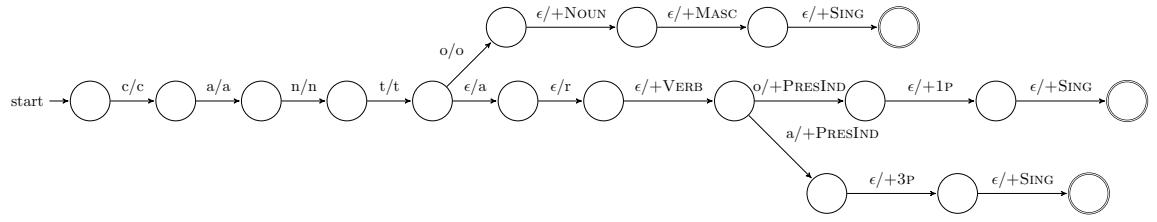


Figure 9.7: Fragment of a finite state transducer for Spanish morphology. There are two accepting paths for the input *canto*: *canto+Noun+Masc+Sing* (masculine singular noun, meaning a song), and *cantar+Verb+PresInd+1p+Sing* (I sing). There is also an accepting path for *canta*, with output *cantar+Verb+PresInd+3p+Sing* (he/she sings).

4643 to accept strings or transductions that are valid. For example, a pluralization transducer
 4644 that does not accept *foot/feet* would undergenerate. Suppose we “fix” the transducer to
 4645 accept this example, but as a side effect, it now accepts *boot/beet*; the transducer would
 4646 then be said to overgenerate. A transducer that accepts *foot/foots* but not *foot/feet* would
 4647 both overgenerate and undergenerate.

4648 9.1.4.4 Finite state composition

4649 Designing finite state transducers to capture the full range of morphological phenomena in
 4650 any real language is a huge task. Modularization is a classic computer science approach for
 4651 this situation: decompose a large and unwieldy problem into a set of subproblems, each
 4652 of which will hopefully have a concise solution. Finite state automata can be modularized
 4653 through composition: feeding the output of one transducer T_1 as the input to another
 4654 transducer T_2 , written $T_2 \circ T_1$. Formally, if there exists some y such that $(x, y) \in T_1$
 4655 (meaning that T_1 produces output y on input x), and $(y, z) \in T_2$, then $(x, z) \in (T_2 \circ T_1)$.
 4656 Because finite state transducers are closed under composition, there is guaranteed to be
 4657 a single finite state transducer that $T_3 = T_2 \circ T_1$, which can be constructed as a machine
 4658 with one state for each pair of states in T_1 and T_2 (Mohri et al., 2002).

4659 Example: Morphology and orthography In English morphology, the suffix -ed is added to
 4660 signal the past tense for many verbs: *cook*→*cooked*, *want*→*wanted*, etc. However, English
 4661 orthography dictates that this process cannot produce a spelling with consecutive e’s,
 4662 so that *bake*→*baked*, not *bakeed*. A modular solution is to build separate transducers
 4663 for morphology and orthography. The morphological transducer T_M transduces from
 4664 *bake+PAST* to *bake+ed*, with the + symbol indicating a segment boundary. The input
 4665 alphabet of T_M includes the lexicon of words and the set of morphological features; the
 4666 output alphabet includes the characters a-z and the + boundary marker. Next, an orthographic
 4667 transducer T_O is responsible for the transductions *cook+ed*→*cooked*, and *bake+ed*→
 4668 *baked*. The input alphabet of T_O must be the same as the output alphabet for T_M , and the

4669 output alphabet is simply the characters a-z. The composed transducer ($T_O \circ T_M$) then
 4670 transduces from 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,

$$\delta(q_{m-1,i}, Y_m = j, q_{m,j}) = \log p(w_m, Y_m = j | Y_{m-1} = i) \quad [9.20]$$

$$= \log p(w_m | Y_m = j) + \log \Pr(Y_m = j | Y_{m-1} = i). \quad [9.21]$$

4671 Because there is only one possible transition for each tag Y_m , this WFSA is deterministic.
 4672 The score for any tag sequence $\{y_m\}_{m=1}^M$ is the sum of these log-probabilities, corresponding
 4673 to the total log probability $\log p(\mathbf{w}, \mathbf{y})$. Furthermore, the trellis can be constructed by the
 4674 composition of simpler FSTs.

- 4675 • First, construct a “transition” transducer to represent a bigram probability model
 4676 over tag sequences, T_T . This transducer is almost identical to the n -gram language
 4677 model acceptor in § 9.1.3.1: there is one state for each tag, and the edge weights
 4678 equal to the transition log-probabilities, $\delta(q_i, j, j, q_j) = \log \Pr(Y_m = j | Y_{m-1} = i)$.
 4679 Note that T_T is a transducer, with identical input and output at each arc; this makes
 4680 it possible to compose T_T with other transducers.
- 4681 • Next, construct an “emission” transducer to represent the probability of words given
 4682 tags, T_E . This transducer has only a single state, with arcs for each word/tag pair,
 4683 $\delta(q_0, i, j, q_0) = \log \Pr(W_m = j | Y_m = i)$. The input vocabulary is the set of all tags,
 4684 and the output vocabulary is the set of all words.
- 4685 • The composition $T_E \circ T_T$ is a finite state transducer with one state per tag, as shown
 4686 in Figure 9.8. Each state has $V \times K$ outgoing edges, representing transitions to each
 4687 of the K other states, with outputs for each of the V words in the vocabulary. The
 4688 weights for these edges are equal to,

$$\delta(q_i, Y_m = j, w_m, q_j) = \log p(w_m, Y_m = j | Y_{m-1} = i). \quad [9.22]$$

- 4689 • The trellis is a structure with $M \times K$ nodes, for each of the M words to be tagged
 4690 and each of the K tags in the tagset. It can be built by composition of $(T_E \circ T_T)$
 4691 against an unweighted chain FSA $M_A(\mathbf{w})$ that is specially constructed to accept only
 4692 a given input w_1, w_2, \dots, w_M , shown in Figure 9.9. The trellis for input \mathbf{w} is built
 4693 from the composition $M_A(\mathbf{w}) \circ (T_E \circ T_T)$. Composing with the unweighted $M_A(\mathbf{w})$
 4694 does not affect the edge weights from $(T_E \circ T_T)$, but it selects the subset of paths
 4695 that generate the word sequence \mathbf{w} .

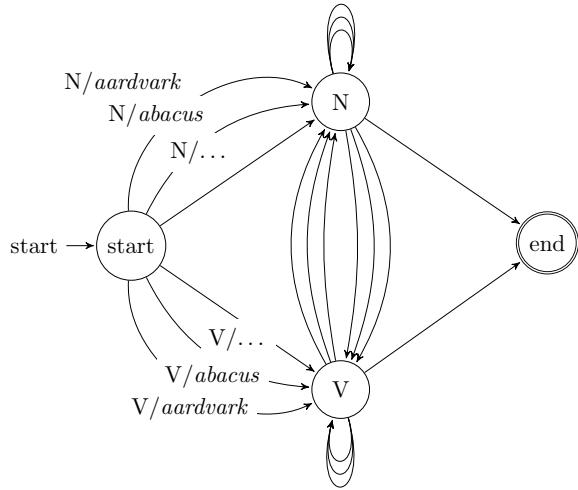


Figure 9.8: Finite state transducer for hidden Markov models, with a small tagset of nouns and verbs. For each pair of tags (including self-loops), there is an edge for every word in the vocabulary. For simplicity, input and output are only shown for the edges from the start state. Weights are also omitted 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 $\log \Pr(Y_m = \diamond \mid Y_{m-1} = i)$.

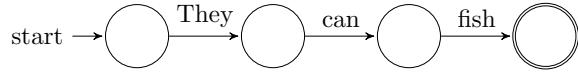


Figure 9.9: Chain finite state acceptor for the input They can fish.

4696 9.1.5 *Learning weighted finite state automata

4697 In generative models such as n -gram language models and hidden Markov models, the
 4698 edge weights correspond to log probabilities, which can be obtained from relative frequency
 4699 estimation. However, in other cases, we wish to learn the edge weights from input/output
 4700 pairs. This is difficult in non-deterministic finite state automata, because we do not observe
 4701 the specific arcs that are traversed in accepting the input, or in transducing from input to
 4702 output. The path through the automaton is a latent variable.

4703 Chapter 5 presented one method for learning with latent variables: expectation maximization
 4704 (EM). This involves computing a distribution $q(\cdot)$ over the latent variable, and iterating
 4705 between updates to this distribution and updates to the parameters — in this case, the
 4706 arc weights. The forward-backward algorithm (§ 7.5.3.3) describes a dynamic program
 4707 for computing a distribution over arcs in the trellis structure of a hidden Markov model,

4708 but this is a special case of the more general problem for finite state automata. Eisner
 4709 (2002) describes an expectation semiring, which enables the expected number of transitions
 4710 across each arc to be computed through a semiring shortest-path algorithm. Alternative
 4711 approaches for generative models include Markov Chain Monte Carlo (Chiang et al., 2010)
 4712 and spectral learning (Balle et al., 2011).

4713 Further afield, we can take a perceptron-style approach, with each arc corresponding
 4714 to a feature. The classic perceptron update would update the weights by subtracting the
 4715 difference between the feature vector corresponding to the predicted path and the feature
 4716 vector corresponding to the correct path. Since the path is not observed, we resort to a
 4717 hidden variable perceptron. The model is described formally in § 12.4, but the basic idea
 4718 is to compute an update from the difference between the features from the predicted path
 4719 and the features for the best-scoring path that generates the correct output.

4720 9.2 Context-free languages

4721 Beyond the class of regular languages lie the context-free languages. An example of a
 4722 language that is context-free but not finite state is the set of arithmetic expressions with
 4723 balanced parentheses. Intuitively, to accept only strings in this language, an FSA would
 4724 have to “count” the number of left parentheses, and make sure that they are balanced
 4725 against the number of right parentheses. An arithmetic expression can be arbitrarily long,
 4726 yet by definition an FSA has a finite number of states. Thus, for any FSA, there will be
 4727 a string that with too many parentheses to count. More formally, the pumping lemma is
 4728 a proof technique for showing that languages are not regular. It is typically demonstrated
 4729 for the simpler case $a^n b^n$, the language of strings containing a sequence of a 's, and then an
 4730 equal-length sequence of b 's.⁴

4731 There are at least two arguments for the relevance of non-regular formal languages to
 4732 linguistics. First, there are natural language phenomena that are argued to be isomorphic
 4733 to $a^n b^n$. For English, the classic example is center embedding, shown in Figure 9.10.
 4734 The initial expression the dog specifies a single dog. Embedding this expression into
 4735 the cat ____ chased specifies a particular cat — the one chased by the dog. This cat can
 4736 then be embedded again to specify a goat, in the less felicitous but arguably grammatical
 4737 expression, the goat the cat the dog chased kissed, which refers to the goat who was kissed
 4738 by the cat which was chased by the dog. Chomsky (1957) argues that to be grammatical,
 4739 a center-embedded construction must be balanced: if it contains n noun phrases (e.g., the
 4740 cat), they must be followed by exactly $n - 1$ verbs. An FSA that could recognize such
 4741 expressions would also be capable of recognizing the language $a^n b^n$. Because we can prove
 4742 that no FSA exists for $a^n b^n$, no FSA can exist for center embedded constructions either.

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

4743 English includes center embedding, and so the argument goes, English grammar as a whole
 4744 cannot be regular.⁵

4745 A more practical argument for moving beyond regular languages is modularity. Many
 4746 linguistic phenomena — especially in syntax — involve constraints that apply at long
 4747 distance. Consider the problem of determiner-noun number agreement in English: we can
 4748 say the coffee and these coffees, but not *these coffee. By itself, this is easy enough to
 4749 model in an FSA. However, fairly complex modifying expressions can be inserted between
 4750 the determiner and the noun:

- 4751 (9.5) the burnt coffee
- 4752 (9.6) the badly-ground coffee
- 4753 (9.7) the burnt and badly-ground Italian coffee
- 4754 (9.8) these burnt and badly-ground Italian coffees
- 4755 (9.9) *these burnt and badly-ground Italian coffee

4756 Again, an FSA can be designed to accept modifying expressions such as burnt and badly-ground
 4757 Italian. Let's call this FSA F_M . To reject the final example, a finite state acceptor must
 4758 somehow "remember" that the determiner was plural when it reaches the noun coffee at
 4759 the end of the expression. The only way to do this is to make two identical copies of F_M :
 4760 one for singular determiners, and one for plurals. While this is possible in the finite state
 4761 framework, it is inconvenient — especially in languages where more than one attribute of
 4762 the noun is marked by the determiner. Context-free languages facilitate modularity across
 4763 such long-range dependencies.

4764 9.2.1 Context-free grammars

4765 Context-free languages are specified by context-free grammars (CFGs), which are tuples
 4766 (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

- 4767 • a finite set of non-terminals N ;
- 4768 • a finite alphabet Σ of terminal symbols;
- 4769 • a set of production rules R , each of the form $A \rightarrow \beta$, where $A \in N$ and $\beta \in (\Sigma \cup N)^*$;
- 4770 • a designated start symbol S .

4771 In the production rule $A \rightarrow \beta$, the left-hand side (LHS) A must be a non-terminal;
 4772 the right-hand side (RHS) can be a sequence of terminals or non-terminals, $\{n, \sigma\}^*, n \in$
 4773 $N, \sigma \in \Sigma$. A non-terminal can appear on the left-hand side of many production rules.
 4774 A non-terminal can appear on both the left-hand side and the right-hand side; this is a
 4775 recursive production, and is analogous to self-loops in finite state automata. The name
 4776 “context-free” is based on the property that the production rule depends only on the LHS,
 4777 and not on its ancestors or neighbors; this is analogous to Markov property of finite state
 4778 automata, in which the behavior at each step depends only on the current state, on not on
 4779 the path by which that state was reached.

4780 A derivation τ is a sequence of steps from the start symbol S to a surface string $w \in \Sigma^*$,
 4781 which is the yield of the derivation. A string w is in a context-free language if there is
 4782 some derivation from S yielding w . Parsing is the problem of finding a derivation for a
 4783 string in a grammar. Algorithms for parsing are described in chapter 10.

4784 Like regular expressions, context-free grammars define the language but not the computation
 4785 necessary to recognize it. The context-free analogues to finite state acceptors are pushdown
 4786 automata, a theoretical model of computation in which input symbols can be pushed onto
 4787 a stack with potentially infinite depth. For more details, see Sipser (2012).

4788 9.2.1.1 Example

4789 Figure 9.11 shows a context-free grammar for arithmetic expressions such as $1 + 2 \div 3 - 4$.
 4790 In this grammar, the terminal symbols include the digits $\{1, 2, \dots, 9\}$ and the operators
 4791 $\{+, -, \times, \div\}$. The rules include the $|$ symbol, a notational convenience that makes it
 4792 possible to specify multiple right-hand sides on a single line: the statement $A \rightarrow x | y$

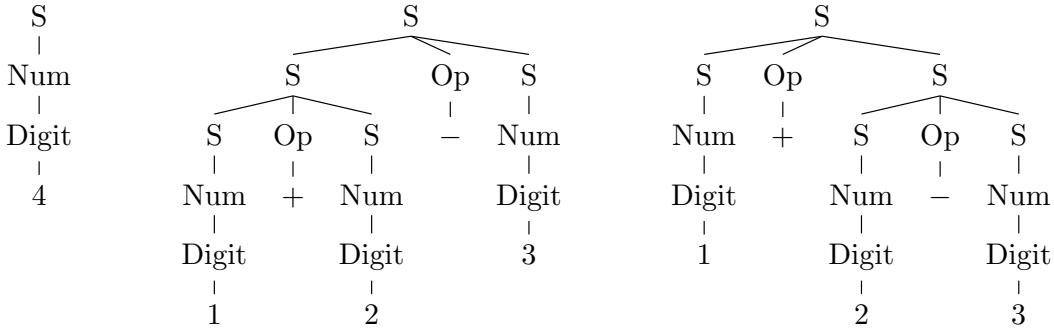


Figure 9.12: Some example derivations from the arithmetic grammar in Figure 9.11

4793 defines two productions, $A \rightarrow x$ and $A \rightarrow y$. This grammar is recursive: the non-termals
4794 S and Num can produce themselves.

4795 Derivations are typically shown as trees, with production rules applied from the top
4796 to the bottom. The tree on the left in Figure 9.12 describes the derivation of a single
4797 digit, through the sequence of productions $S \rightarrow \text{Num} \rightarrow \text{Digit} \rightarrow 4$ (these are all unary
4798 productions, because the right-hand side contains a single element). The other two trees in
4799 Figure 9.12 show alternative derivations of the string $1 + 2 - 3$. The existence of multiple
4800 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:

$$(S (S (S (\text{Num} (\text{Digit } 1))) (\text{Op } +) (S (\text{Num} (\text{Digit } 2)))) (\text{Op } -) (S (\text{Num} (\text{Digit } 3)))) \quad [9.23]$$

$$(S (S (\text{Num} (\text{Digit } 1))) (\text{Op } +) (S (\text{Num} (\text{Digit } 2)) (\text{Op } -) (S (\text{Num} (\text{Digit } 3))))). \quad [9.24]$$

4801 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 $n > 0$.

$$\begin{aligned} S &\rightarrow aSb \mid ab \\ S &\rightarrow aSb \mid aabb \mid ab \end{aligned}$$

4802 Two grammars are weakly equivalent if they generate the same strings. Two grammars
4803 are strongly equivalent if they generate the same strings via the same derivations. The
4804 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:

$$A \rightarrow BC$$

$$A \rightarrow a$$

- 4805 All CFGs can be converted into a CNF grammar that is weakly equivalent. To convert a
 4806 grammar into CNF, we first address productions that have more than two non-terminals on
 4807 the RHS by creating new “dummy” non-terminals. For example, if we have the production,

$$W \rightarrow X Y Z, \quad [9.25]$$

it is replaced with two productions,

$$W \rightarrow X W \setminus X \quad [9.26]$$

$$W \setminus X \rightarrow Y Z. \quad [9.27]$$

- 4808 In these productions, $W \setminus X$ is a new dummy non-terminal. This transformation binarizes
 4809 the grammar, which is critical for efficient bottom-up parsing, as we will see in chapter 10.
 4810 Productions whose right-hand side contains a mix of terminal and non-terminal symbols
 4811 can be replaced in a similar fashion.

- 4812 Unary non-terminal productions $A \rightarrow B$ are replaced as follows: identify all productions
 4813 $B \rightarrow \alpha$, and add $A \rightarrow \alpha$ to the grammar. For example, in the grammar described in
 4814 Figure 9.11, we would replace $\text{Num} \rightarrow \text{Digit}$ with $\text{Num} \rightarrow 1 | 2 | \dots | 9$. However, we keep
 4815 the production $\text{Num} \rightarrow \text{Num Digit}$, which is a valid binary production.

4816 9.2.2 Natural language syntax as a context-free language

- 4817 Context-free grammars are widely used to represent syntax, which is the set of rules that
 4818 determine whether an utterance is judged to be grammatical. If this representation were
 4819 perfectly faithful, then a natural language such as English could be transformed into a
 4820 formal language, consisting of exactly the (infinite) set of strings that would be judged to
 4821 be grammatical by a fluent English speaker. We could then build parsing software that
 4822 would automatically determine if a given utterance were grammatical.⁷

- 4823 Contemporary theories generally do not consider natural languages to be context-free
 4824 (see § 9.3), yet context-free grammars are widely used in natural language parsing. The
 4825 reason is that context-free representations strike a good balance: they cover a broad
 4826 range of syntactic phenomena, and they can be parsed efficiently. This section therefore
 4827 describes 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).

4828 the conventions of the Penn Treebank (PTB; Marcus et al., 1993), a large-scale annotation
 4829 of English language syntax. The generalization to “mildly” context-sensitive languages is
 4830 discussed in § 9.3.

4831 The Penn Treebank annotation is a phrase-structure grammar of English. This means
 4832 that sentences are broken down into constituents, which are contiguous sequences of words
 4833 that function as coherent units for the purpose of linguistic analysis. Constituents generally
 4834 have a few key properties:

4835 Movement. Constituents can often be moved around sentences as units.

4836 (9.10) Abigail gave (her brother) (a fish).

4837 (9.11) Abigail gave (a fish) to (her brother).

4838 In contrast, gave her and brother a cannot easily be moved while preserving grammaticality.

4839 Substitution. Constituents can be substituted by other phrases of the same type.

4840 (9.12) Max thanked (his older sister).

4841 (9.13) Max thanked (her).

4842 In contrast, substitution is not possible for other contiguous units like Max thanked
 4843 and thanked his.

4844 Coordination. Coordinators like and and or can conjoin constituents.

4845 (9.14) (Abigail) and (her younger brother) bought a fish.

4846 (9.15) Abigail (bought a fish) and (gave it to Max).

4847 (9.16) Abigail (bought) and (greedily ate) a fish.

4848 Units like brother bought and bought a cannot easily be coordinated.

4849 These examples argue for units such as her brother and bought a fish to be treated as
 4850 constituents. Other sequences of words in these examples, such as Abigail gave and brother
 4851 a fish, cannot be moved, substituted, and coordinated in these ways. In phrase-structure
 4852 grammar, constituents are nested, so that the senator from New Jersey contains the
 4853 constituent from New Jersey, which in turn contains New Jersey. The sentence itself is the
 4854 maximal constituent; each word is a minimal constituent, derived from a unary production
 4855 from a part-of-speech tag. Between part-of-speech tags and sentences are phrases. In
 4856 phrase-structure grammar, phrases have a type that is usually determined by their head
 4857 word: for example, a noun phrase corresponds to a noun and the group of words that

4858 modify it, such as her younger brother; a verb phrase includes the verb and its modifiers,
4859 such as bought a fish and greedily ate it.

4860 In context-free grammars, each phrase type is a non-terminal, and each constituent
4861 is the substring that the non-terminal yields. Grammar design involves choosing the
4862 right set of non-terminals. Fine-grained non-terminals make it possible to represent more
4863 fine-grained linguistic phenomena. For example, by distinguishing singular and plural noun
4864 phrases, it is possible to have a grammar of English that generates only sentences that
4865 obey subject-verb agreement. However, enforcing subject-verb agreement is considerably
4866 more complicated in languages like Spanish, where the verb must agree in both person and
4867 number with subject. In general, grammar designers must trade off between overgeneration
4868 — a grammar that permits ungrammatical sentences — and undergeneration — a grammar
4869 that fails to generate grammatical sentences. Furthermore, if the grammar is to support
4870 manual annotation of syntactic structure, it must be simple enough to annotate efficiently.

4871 9.2.3 A phrase-structure grammar for English

4872 To better understand how phrase-structure grammar works, let's consider the specific case
4873 of the Penn Treebank grammar of English. The main phrase categories in the Penn
4874 Treebank (PTB) are based on the main part-of-speech classes: noun phrase (NP), verb
4875 phrase (VP), prepositional phrase (PP), adjectival phrase (ADJP), and adverbial phrase
4876 (ADVP). The top-level category is S, which conveniently stands in for both “sentence”
4877 and the “start” symbol. Complement clauses (e.g., I take the good old fashioned ground
4878 that the whale is a fish) are represented by the non-terminal SBAR. The terminal symbols
4879 in the grammar are individual words, which are generated from unary productions from
4880 part-of-speech tags (the PTB tagset is described in § 8.1).

4881 This section explores the productions from the major phrase-level categories, explaining
4882 how to generate individual tag sequences. The production rules are approached in a
4883 “theory-driven” manner: first the syntactic properties of each phrase type are described,
4884 and then some of the necessary production rules are listed. But it is important to keep in
4885 mind that the Penn Treebank was produced in a “data-driven” manner. After the set of
4886 non-terminals was specified, annotators were free to analyze each sentence in whatever way
4887 seemed most linguistically accurate, subject to some high-level guidelines. The grammar
4888 of the Penn Treebank is simply the set of productions that were required to analyze the
4889 several million words of the corpus. By design, the grammar overgenerates — it does not
4890 exclude ungrammatical sentences.

4891 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:

$$S \rightarrow ADVP \ NP \ VP \quad \text{Unfortunately Abigail ate the kimchi.} \quad [9.29]$$

$$S \rightarrow S \ Cc \ S \quad \text{Abigail ate the kimchi and Max had a burger.} \quad [9.30]$$

$$S \rightarrow VP \quad \text{Eat the kimchi.} \quad [9.31]$$

- 4892 where ADVP is an adverbial phrase (e.g., unfortunately, very unfortunately) and Cc is a
 4893 coordinating conjunction (e.g., and, but).⁸

4894 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:

$$NP \rightarrow Nn \mid Nns \mid Nnp \mid Prp \quad [9.32]$$

$$NP \rightarrow Det \ Nn \mid Det \ Nns \mid Det \ Nnp \quad [9.33]$$

- 4895 The tags Nn, Nns, and Nnp refer to singular, plural, and proper nouns; Prp refers to
 4896 personal pronouns, and Det refers to determiners. The grammar also contains terminal
 4897 productions from each of these tags, e.g., Prp → I | you | we |

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:

$$NP \rightarrow ADJP \ Nn \mid ADJP \ Nns \mid Det \ ADJP \ Nn \mid Det \ ADJP \ Nns \quad [9.34]$$

$$NP \rightarrow Cd \ Nns \mid Det \ Cd \ Nns \mid \dots \quad [9.35]$$

Some noun phrases include multiple nouns, such as the liberation movement and an antelope horn, necessitating additional productions:

$$NP \rightarrow Nn \ Nn \mid Nn \ Nns \mid Det \ Nn \ Nn \mid \dots \quad [9.36]$$

- 4898 These multiple noun constructions can be combined with adjectival phrases and cardinal
 4899 numbers, leading to a large number of additional productions.

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.

Recursive noun phrase productions include coordination, prepositional phrase attachment, subordinate clauses, and verb phrase adjuncts:

$NP \rightarrow NP\ Cc\ NP$	e.g., the red and the black	[9.37]
$NP \rightarrow NP\ PP$	e.g., the President of the Georgia Institute of Technology	[9.38]
$NP \rightarrow NP\ SBAR$	e.g., a whale which he had wounded	[9.39]
$NP \rightarrow NP\ VP$	e.g., a whale taken near Shetland	[9.40]

4900 These recursive productions are a major source of ambiguity, because the VP and PP
 4901 non-terminals can also generate NP children. Thus, the the President of the Georgia
 4902 Institute of Technology can be derived in two ways, as can a whale taken near Shetland in
 4903 October.

4904 But aside from these few recursive productions, the noun phrase fragment of the Penn
 4905 Treebank grammar is relatively flat, containing a large of number of productions that go
 4906 from NP directly to a sequence of parts-of-speech. If noun phrases had more internal
 4907 structure, the grammar would need fewer rules, which, as we will see, would make parsing
 4908 faster and machine learning easier. Vadas and Curran (2011) propose to add additional
 4909 structure in the form of a new non-terminal called a nominal modifier (NML), e.g.,

4910 (9.17) ($NP\ (NN\ crude)\ (NN\ oil)\ (NNS\ prices)$) (PTB analysis)
 4911 ($NP\ (NML\ (NN\ crude)\ (NN\ oil))\ (NNS\ prices)$) (NML-style analysis)

4912 Another proposal is to treat the determiner as the head of a determiner phrase (DP;
 4913 Abney, 1987). There are linguistic arguments for and against determiner phrases (e.g.,
 4914 Van Eynde, 2006). From the perspective of context-free grammar, DPs enable more
 4915 structured analyses of some constituents, e.g.,

4916 (9.18) ($NP\ (DT\ the)\ (JJ\ white)\ (NN\ whale)$) (PTB analysis)
 4917 ($DP\ (DT\ the)\ (NP\ (JJ\ white)\ (NN\ whale))$) (DP-style analysis).

4918 9.2.3.3 Verb phrases

Verb phrases describe actions, events, and states of being. The PTB tagset distinguishes several classes of verb inflections: base form (Vb; she likes to snack), present-tense third-person singular (Vbz; she snacks), present tense but not third-person singular (Vbp; they snack), past tense (Vbd; they snacked), present participle (Vbg; they are snacking), and past participle (Vbn; they had snacked).⁹ Each of these forms can constitute a verb phrase on its own:

$VP \rightarrow Vb \mid Vbz \mid Vbd \mid Vbn \mid Vbg \mid Vbp$ [9.41]

⁹This tagset is specific to English: for example, Vbp is a meaningful category only because English morphology distinguishes third-person singular from all person-number combinations.

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):

$VP \rightarrow Md\ VP$	She will snack	[9.42]
$VP \rightarrow Vbd\ VP$	She had snacked	[9.43]
$VP \rightarrow Vbz\ VP$	She has been snacking	[9.44]
$VP \rightarrow Vbn\ VP$	She has been snacking	[9.45]
$VP \rightarrow To\ VP$	She wants to snack	[9.46]
$VP \rightarrow VP\ Cc\ VP$	She buys and eats many snacks	[9.47]

- 4919 Each of these productions uses recursion, with the VP non-terminal appearing in both the
 4920 LHS and RHS. This enables the creation of complex verb phrases, such as She will have
 4921 wanted to have been snacking.

Transitive verbs take noun phrases as direct objects, and ditransitive verbs take two direct objects:

$VP \rightarrow Vbz\ NP$	She teaches algebra	[9.48]
$VP \rightarrow Vbg\ NP$	She has been teaching algebra	[9.49]
$VP \rightarrow Vbd\ NP\ NP$	She taught her brother algebra	[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:

$VP \rightarrow Vbz\ S$	Asha wants to eat the kimchi	[9.51]
$VP \rightarrow Vbz\ SBAR$	Asha knows that Boyang eats the kimchi	[9.52]

- 4922 The first production overgenerates, licensing sentences like *Asha sees Boyang eats the
 4923 kimchi. This problem could be addressed by designing a more specific set of sentence
 4924 non-terminals, indicating whether the main verb can be conjugated.

Verbs can also be modified by prepositional phrases and adverbial phrases:

$VP \rightarrow Vbz\ PP$	She studies at night	[9.53]
$VP \rightarrow Vbz\ ADVP$	She studies intensively	[9.54]
$VP \rightarrow ADVP\ Vbg$	She is not studying	[9.55]

- 4925 Again, because these productions are not recursive, the grammar must include productions
 4926 for every verb part-of-speech.

A special set of verbs, known as copula, can take predicative adjectives as direct objects:

$VP \rightarrow Vbz\ ADJP$	She is hungry	[9.56]
----------------------------	---------------	--------

$VP \rightarrow Vbp\ ADJP$	Success seems increasingly unlikely	[9.57]
----------------------------	-------------------------------------	--------

- 4927 The PTB does not have a special non-terminal for copular verbs, so this production
 4928 generates non-grammatical examples such as *She eats tall.

Particles (PRT as a phrase; Rp as a part-of-speech) work to create phrasal verbs:

$VP \rightarrow Vb\ PRT$	She told them to fuck off	[9.58]
--------------------------	---------------------------	--------

$VP \rightarrow Vbd\ PRT\ NP$	They gave up their ill-gotten gains	[9.59]
-------------------------------	-------------------------------------	--------

- 4929 As the second production shows, particle productions are required for all configurations of
 4930 verb parts-of-speech and direct objects.

4931 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 → 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]
-----------------------------	--	--------

- 4932 The tag Rbr is a comparative adverb.

Adjectival phrases extend beyond bare adjectives (ADJP → Jj) in a number of ways:

$ADJP \rightarrow Rb\ Jj$	very hungry	[9.66]
---------------------------	-------------	--------

$ADJP \rightarrow Rbr\ Jj$	more hungry	[9.67]
----------------------------	-------------	--------

$ADJP \rightarrow Jjs\ Jj$	best possible	[9.68]
----------------------------	---------------	--------

$ADJP \rightarrow Rb\ Jjr$	even bigger	[9.69]
----------------------------	-------------	--------

$ADJP \rightarrow Jj\ Cc\ Jj$	high and mighty	[9.70]
-------------------------------	-----------------	--------

$ADJP \rightarrow Jj\ Jj$	West German	[9.71]
---------------------------	-------------	--------

$ADJP \rightarrow Rb\ Vbn$	previously reported	[9.72]
----------------------------	---------------------	--------

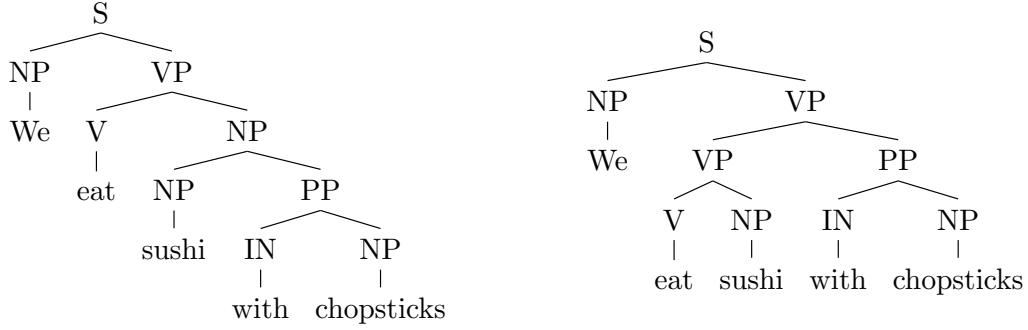


Figure 9.13: Two derivations of the same sentence

4933 The tags Jjr and Jjs refer to comparative and superlative adjectives respectively.

All of these phrase types can be coordinated:

$PP \rightarrow PP\ Cc\ PP$	on time and under budget	[9.73]
$ADVP \rightarrow ADVP\ Cc\ ADVP$	now and two years ago	[9.74]
$ADJP \rightarrow ADJP\ Cc\ ADJP$	quaint and rather deceptive	[9.75]
$SBAR \rightarrow SBAR\ Cc\ SBAR$	whether they want control	[9.76]
	or whether they want exports	

4934 9.2.4 Grammatical ambiguity

4935 Context-free parsing is useful not only because it determines whether a sentence is grammatical,
 4936 but mainly because the constituents and their relations can be applied to tasks such as
 4937 information extraction (chapter 17) and sentence compression (Jing, 2000; Clarke and
 4938 Lapata, 2008). However, the ambiguity of wide-coverage natural language grammars poses
 4939 a serious problem for such potential applications. As an example, Figure 9.13 shows
 4940 two possible analyses for the simple sentence We eat sushi with chopsticks, depending
 4941 on whether the chopsticks modify eat or sushi. Realistic grammars can license thousands
 4942 or even millions of parses for individual sentences. Weighted context-free grammars solve
 4943 this problem by attaching weights to each production, and selecting the derivation with
 4944 the highest score. This is the focus of chapter 10.

4945 9.3 *Mildly context-sensitive languages

4946 Beyond context-free languages lie context-sensitive languages, in which the expansion
 4947 of a non-terminal depends on its neighbors. In the general class of context-sensitive
 4948 languages, computation becomes much more challenging: the membership problem for

4949 context-sensitive languages is PSPACE-complete. Since PSPACE contains the complexity
 4950 class NP (problems that can be solved in polynomial time on a non-deterministic Turing
 4951 machine), PSPACE-complete problems cannot be solved efficiently if $P \neq NP$. Thus,
 4952 designing an efficient parsing algorithm for the full class of context-sensitive languages
 4953 is probably hopeless.¹⁰

4954 However, Joshi (1985) identifies a set of properties that define mildly context-sensitive
 4955 languages, which are a strict subset of context-sensitive languages. Like context-free
 4956 languages, mildly context-sensitive languages are efficiently parseable. However, the mildly
 4957 context-sensitive languages include non-context-free languages, such as the “copy language”
 4958 $\{ww \mid w \in \Sigma^*\}$ and the language $a^m b^n c^m d^n$. Both are characterized by cross-serial
 4959 dependencies, linking symbols at long distance across the string.¹¹ For example, in the
 4960 language $a^n b^m c^n d^m$, each a symbol is linked to exactly one c symbol, regardless of the
 4961 number of intervening b symbols.

4962 9.3.1 Context-sensitive phenomena in natural language

4963 Such phenomena are occasionally relevant to natural language. A classic example is found
 4964 in Swiss-German (Shieber, 1985), in which sentences such as we let the children help Hans
 4965 paint the house are realized by listing all nouns before all verbs, i.e., we the children Hans
 4966 the house let help paint. Furthermore, each noun’s determiner is dictated by the noun’s
 4967 case marking (the role it plays with respect to the verb). Using an argument that is
 4968 analogous to the earlier discussion of center-embedding (§ 9.2), Shieber argues that these
 4969 case marking constraints are a cross-serial dependency, homomorphic to $a^m b^n c^m d^n$, and
 4970 therefore not context-free.

4971 As with the move from regular to context-free languages, mildly context-sensitive
 4972 languages can be motivated by expedience. While infinite sequences of cross-serial dependencies
 4973 cannot be handled by context-free grammars, even finite sequences of cross-serial dependencies
 4974 are more convenient to handle using a mildly context-sensitive formalism like tree-adjoining
 4975 grammar (TAG) and combinatory categorial grammar (CCG). Furthermore, TAG-inspired
 4976 parsers have been shown to be particularly effective in parsing the Penn Treebank (Collins,
 4977 1997; Carreras et al., 2008), and CCG plays a leading role in current research on semantic
 4978 parsing (Zettlemoyer and Collins, 2005). Furthermore, these two formalisms are weakly
 4979 equivalent: any language that can be specified in TAG can also be specified in CCG, and
 4980 vice versa (Joshi et al., 1991). The remainder of the chapter gives a brief overview of CCG,

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).

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 $\{a^{2^n} \mid n \geq 0\}$.

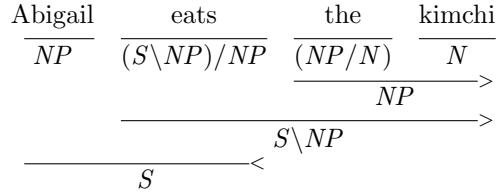


Figure 9.14: A syntactic analysis in CCG involving forward and backward function application

4981 but you are encouraged to consult Joshi and Schabes (1997) and Steedman and Baldridge
 4982 (2011) for more detail on TAG and CCG respectively.

4983 9.3.2 Combinatory categorial grammar

4984 In combinatory categorial grammar, structural analyses are built up through a small set
 4985 of generic combinatorial operations, which apply to immediately adjacent sub-structures.
 4986 These operations act on the categories of the sub-structures, producing a new structure
 4987 with a new category. The basic categories include S (sentence), NP (noun phrase), VP
 4988 (verb phrase) and N (noun). The goal is to label the entire span of text as a sentence, S.

4989 Complex categories, or types, are constructed from the basic categories, parentheses,
 4990 and forward and backward slashes: for example, S/NP is a complex type, indicating a
 4991 sentence that is lacking a noun phrase to its right; $S\NP$ is a sentence lacking a noun phrase
 4992 to its left. Complex types act as functions, and the most basic combinatory operations are
 4993 function application to either the right or left neighbor. For example, the type of a verb
 4994 phrase, such as eats, would be $S\NP$. Applying this function to a subject noun phrase to
 4995 its left results in an analysis of Abigail eats as category S, indicating a successful parse.

4996 Transitive verbs must first be applied to the direct object, which in English appears to
 4997 the right of the verb, before the subject, which appears on the left. They therefore have the
 4998 more complex type $(S\NP)/NP$. Similarly, the application of a determiner to the noun at
 4999 its right results in a noun phrase, so determiners have the type NP/N . Figure 9.14 provides
 5000 an example involving a transitive verb and a determiner. A key point from this example
 5001 is that it can be trivially transformed into phrase-structure tree, by treating each function
 5002 application as a constituent phrase. Indeed, when CCG's only combinatory operators
 5003 are forward and backward function application, it is equivalent to context-free grammar.
 5004 However, the location of the “effort” has changed. Rather than designing good productions,
 5005 the grammar designer must focus on the lexicon — choosing the right categories for each
 5006 word. This makes it possible to parse a wide range of sentences using only a few generic
 5007 combinatory operators.

5008 Things become more interesting with the introduction of two additional operators:
 5009 composition and type-raising. Function composition enables the combination of complex

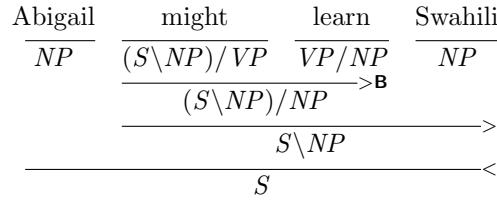


Figure 9.15: A syntactic analysis in CCG involving function composition (example modified from Steedman and Baldridge, 2011)

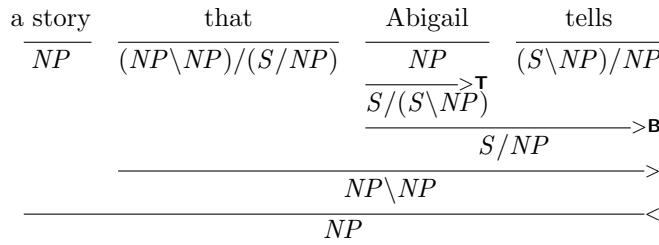


Figure 9.16: A syntactic analysis in CCG involving an object relative clause (based on slides from Alex Clark)

5010 types: $X/Y \circ Y/Z \Rightarrow_{\mathbf{B}} X/Z$ (forward composition) and $Y\backslash Z \circ X\backslash Y \Rightarrow_{\mathbf{B}} X\backslash Z$ (backward
 5011 composition).¹² Composition makes it possible to “look inside” complex types, and combine
 5012 two adjacent units if the “input” for one is the “output” for the other. Figure 9.15 shows
 5013 how function composition can be used to handle modal verbs. While this sentence can
 5014 be parsed using only function application, the composition-based analysis is preferable
 5015 because the unit might learn functions just like a transitive verb, as in the example Abigail
 5016 studies Swahili. This in turn makes it possible to analyze conjunctions such as Abigail
 5017 studies and might learn Swahili, attaching the direct object Swahili to the entire conjoined
 5018 verb phrase studies and might learn. The Penn Treebank grammar fragment from § 9.2.3
 5019 would be unable to handle this case correctly: the direct object Swahili could attach only
 5020 to the second verb learn.

5021 Type raising converts an element of type X to a more complex type: $X \Rightarrow_T T/(T\backslash X)$
 5022 (forward type-raising to type T), and $X \Rightarrow_T T\backslash(T/X)$ (backward type-raising to type
 5023 T). Type-raising makes it possible to reverse the relationship between a function and its
 5024 argument — by transforming the argument into a function over functions over arguments!
 5025 An example may help. Figure 9.15 shows how to analyze an object relative clause, a story
 5026 that Abigail tells. The problem is that tells is a transitive verb, expecting a direct object to
 5027 its right. As a result, Abigail tells is not a valid constituent. The issue is resolved by raising
 5028 Abigail from NP to the complex type $(S/NP)\backslash NP$. This function can then be combined

¹²The subscript **B** follows notation from Curry and Feys (1958).

5029 with the transitive verb tells by forward composition, resulting in the type (S/NP), which
 5030 is a sentence lacking a direct object to its right.¹³ From here, we need only design the
 5031 lexical entry for the complementizer that to expect a right neighbor of type (S/NP), and
 5032 the remainder of the derivation can proceed by function application.

5033 Composition and type-raising give CCG considerable power and flexibility, but at a
 5034 price. The simple sentence Abigail tells Max can be parsed in two different ways: by
 5035 function application (first forming the verb phrase tells Max), and by type-raising and
 5036 composition (first forming the non-constituent Abigail tells). This derivational ambiguity
 5037 does not affect the resulting linguistic analysis, so it is sometimes known as spurious
 5038 ambiguity. Hockenmaier and Steedman (2007) present a translation algorithm for converting
 5039 the Penn Treebank into CCG derivations, using composition and type-raising only when
 5040 necessary.

5041 Exercises

- 5042 1. Sketch out the state diagram for finite-state acceptors for the following languages on
 5043 the alphabet $\{a, b\}$.
 - 5044 a) Even-length strings. (Be sure to include 0 as an even number.)
 - 5045 b) Strings that contain aaa as a substring.
 - 5046 c) Strings containing an even number of a and an odd number of b symbols.
 - 5047 d) Strings in which the substring bbb must be terminal if it appears — the string
 5048 need not contain bbb , but if it does, nothing can come after it.
- 5049 2. Levenshtein edit distance is the number of insertions, substitutions, or deletions
 5050 required to convert one string to another.
 - 5051 a) Define a finite-state acceptor that accepts all strings with edit distance 1 from
 5052 the target string, target.
 - 5053 b) Now think about how to generalize your design to accept all strings with edit
 5054 distance from the target string equal to d . If the target string has length ℓ , what
 5055 is the minimal number of states required?
- 5056 3. Construct an FSA in the style of Figure 9.3, which handles the following examples:
 - 5057 • nation/N, national/Adj, nationalize/V, nationalizer/N
 - 5058 • America/N, American/Adj, Americanize/V, Americanizer/N

¹³The missing direct object would be analyzed as a trace in CFG-like approaches to syntax, including the Penn Treebank.

5059 Be sure that your FSA does not accept any further derivations, such as *nationalizeral
 5060 and *Americanizern.

- 5061 4. Show how to construct a trigram language model in a weighted finite-state acceptor.
 5062 Make sure that you handle the edge cases at the beginning and end of the sequence
 5063 accurately.
- 5064 5. Extend the FST in Figure 9.6 to handle the other two parts of rule 1a of the Porter
 5065 stemmer: -sses → ss, and -ies → -i.
- 5066 6. § 9.1.4.4 describes T_O , a transducer that captures English orthography by transducing
 5067 cook + ed → cooked and bake + ed → baked. Design an unweighted finite-state
 5068 transducer that captures this property of English orthography.

5069 Next, augment the transducer to appropriately model the suffix -s when applied to
 5070 words ending in s, e.g. kiss+s → kisses.

- 5071 7. Add parenthesization to the grammar in Figure 9.11 so that it is no longer ambiguous.
 5072
- 5073 8. Construct three examples — a noun phrase, a verb phrase, and a sentence — which
 5074 can be derived from the Penn Treebank grammar fragment in § 9.2.3, yet are not
 5075 grammatical. Avoid reusing examples from the text. Optionally, propose corrections
 5076 to the grammar to avoid generating these cases.
- 5077 9. Produce parses for the following sentences, using the Penn Treebank grammar fragment
 5078 from § 9.2.3.

- 5079 (9.19) This aggression will not stand.
 5080 (9.20) I can get you a toe.
 5081 (9.21) Sometimes you eat the bar and sometimes the bar eats you.

5082 Then produce parses for three short sentences from a news article from this week.

- 5083 10. * One advantage of CCG is its flexibility in handling coordination:

- 5084 (9.22) Abigail and Max speak Swahili
 5085 (9.23) Abigail speaks and Max understands Swahili

Define the lexical entry for and as

$$\text{and} := (X/X) \setminus X, \quad [9.77]$$

5086 where X can refer to any type. Using this lexical entry, show how to parse the
5087 two examples above. In the second example, Swahili should be combined with
5088 the coordination Abigail speaks and Max understands, and not just with the verb
5089 understands.

5090 Chapter 10

5091 Context-free parsing

5092 Parsing is the task of determining whether a string can be derived from a given context-free
5093 grammar, and if so, how. The parse structure can answer basic questions of who-did-what-to-whom,
5094 and is useful for various downstream tasks, such as semantic analysis (chapter 12 and 13)
5095 and information extraction (chapter 17).

For a given input and grammar, how many parse trees are there? Consider a minimal context-free grammar with only one non-terminal, X, and the following productions:

$$\begin{aligned} X \rightarrow & X \ X \\ X \rightarrow & \text{aardvark} \mid \text{abacus} \mid \dots \mid \text{zyther} \end{aligned}$$

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.,

$$(((w_1 w_2) w_3) w_4) w_5), \quad (((w_1 (w_2 w_3)) w_4) w_5), \quad ((w_1 (w_2 (w_3 w_4))) w_5), \quad \dots$$

5096 The number of such bracketings is a Catalan number, which grows super-exponentially in
5097 the length of the sentence, $C_n = \frac{(2n)!}{(n+1)!n!}$. As with sequence labeling, it is only possible to
5098 exhaustively search the space of parses by resorting to locality assumptions, which make it
5099 possible to search efficiently by reusing shared substructures with dynamic programming.
5100 This chapter focuses on a bottom-up dynamic programming algorithm, which enables
5101 exhaustive search of the space of possible parses, but imposes strict limitations on the
5102 form of scoring function. These limitations can be relaxed by abandoning exhaustive
5103 search. Non-exact search methods will be briefly discussed at the end of this chapter, and
5104 one of them — transition-based parsing — will be the focus of chapter 11.

S	\rightarrow	NP VP
NP	\rightarrow	NP PP we sushi chopsticks
PP	\rightarrow	In NP
In	\rightarrow	with
VP	\rightarrow	V NP VP PP
V	\rightarrow	eat

Table 10.1: A toy example context-free grammar

5105 10.1 Deterministic bottom-up parsing

5106 The CKY algorithm¹ is a bottom-up approach to parsing in a context-free grammar. It
 5107 efficiently tests whether a string is in a language, without enumerating all possible parses.
 5108 The algorithm first forms small constituents, and then tries to merge them into larger
 5109 constituents.

5110 To understand the algorithm, consider the input, We eat sushi with chopsticks. According
 5111 to the toy grammar in Table 10.1, each terminal symbol can be generated by exactly one
 5112 unary production, resulting in the sequence NP V NP In NP. The next step is to try to
 5113 apply binary productions to merge adjacent symbols into larger constituents: for example,
 5114 V NP can be merged into a verb phrase (VP), and In NP can be merged into a prepositional
 5115 phrase (PP). Bottom-up parsing searches for a series of mergers that ultimately results in
 5116 the start symbol S covering the entire input.

5117 The CKY algorithm systematizes this search by incrementally constructing a table t in
 5118 which each cell $t[i, j]$ contains the set of nonterminals that can derive the span $w_{i+1:j}$. The
 5119 algorithm fills in the upper right triangle of the table; it begins with the diagonal, which
 5120 corresponds to substrings of length 1, and then computes derivations for progressively
 5121 larger substrings, until reaching the upper right corner $t[0, M]$, which corresponds to the
 5122 entire input, $w_{1:M}$. If the start symbol S is in $t[0, M]$, then the string w is in the language
 5123 defined by the grammar. This process is detailed in Algorithm 13, and the resulting data
 5124 structure is shown in Figure 10.1. Informally, here's how it works:

- 5125 • Begin by filling in the diagonal: the cells $t[m - 1, m]$ for all $m \in \{1, 2, \dots, M\}$. These
 5126 cells are filled with terminal productions that yield the individual tokens; for the
 5127 word $w_2 = \text{sushi}$, we fill in $t[1, 2] = \{\text{NP}\}$, and so on.
- 5128 • Then fill in the next diagonal, in which each cell corresponds to a subsequence of
 5129 length two: $t[0, 2], t[1, 3], \dots, t[M-2, M]$. These cells are filled in by looking for binary
 5130 productions capable of producing at least one entry in each of the cells corresponding

¹The name is for Cocke-Kasami-Younger, the inventors of the algorithm. It is a special case chart parsing, because its stores reusable computations in a chart-like data structure.

Algorithm 13 The CKY algorithm for parsing a sequence $w \in \Sigma^*$ in a context-free grammar $G = (N, \Sigma, R, S)$, with non-terminals N , production rules R , and start symbol S . The grammar is assumed to be in Chomsky normal form (§ 9.2.1.2). The function $\text{PickFrom}(b[i, j, X])$ selects an element of the set $b[i, j, X]$ arbitrarily. All values of t and b are initialized to \emptyset .

```

1: procedure CKY( $w, G = (N, \Sigma, R, S)$ )
2:   for  $m \in \{1 \dots M\}$  do
3:      $t[m - 1, m] \leftarrow \{X : (X \rightarrow w_m) \in R\}$ 
4:   for  $\ell \in \{2, 3, \dots, M\}$  do                                ▷ Iterate over constituent lengths
5:     for  $m \in \{0, 1, \dots, M - \ell\}$  do                      ▷ Iterate over left endpoints
6:       for  $k \in \{m + 1, m + 2, \dots, m + \ell - 1\}$  do          ▷ Iterate over split points
7:         for  $(X \rightarrow Y Z) \in R$  do                          ▷ Iterate over rules
8:           if  $Y \in t[m, k] \wedge Z \in t[k, m + \ell]$  then
9:              $t[m, m + \ell] \leftarrow t[m, m + \ell] \cup X$           ▷ Add non-terminal to table
10:             $b[m, m + \ell, X] \leftarrow b[m, m + \ell, X] \cup (Y, Z, k)$     ▷ Add back-pointers
11:   if  $S \in t[0, M]$  then
12:     return TraceBack( $S, 0, M, b$ )
13:   else
14:     return  $\emptyset$ 
15: procedure TraceBack( $X, i, j, b$ )
16:   if  $j = i + 1$  then
17:     return  $X$ 
18:   else
19:      $(Y, Z, k) \leftarrow \text{PickFrom}(b[i, j, X])$ 
20:     return  $X \rightarrow (\text{TraceBack}(Y, i, k, b), \text{TraceBack}(Z, k, j, b))$ 

```

5131 to left and right children. For example, the cell $t[1, 3]$ includes VP because the
 5132 grammar includes the production $\text{VP} \rightarrow \text{V NP}$, and the chart contains $\text{V} \in t[1, 2]$
 5133 and $\text{NP} \in t[2, 3]$.

- 5134 • At the next diagonal, the entries correspond to spans of length three. At this level,
 5135 there is an additional decision at each cell: where to split the left and right children.
 5136 The cell $t[i, j]$ corresponds to the subsequence $w_{i+1:j}$, and we must choose some split
 5137 point $i < k < j$, so that $w_{i+1:k}$ is the left child and $w_{k+1:j}$ is the right child. We
 5138 consider all possible k , looking for productions that generate elements in $t[i, k]$ and
 5139 $t[k, j]$; the left-hand side of all such productions can be added to $t[i, j]$. When it is
 5140 time to compute $t[i, j]$, the cells $t[i, k]$ and $t[k, j]$ are guaranteed to be complete, since
 5141 these cells correspond to shorter sub-strings of the input.
- 5142 • The process continues until we reach $t[0, M]$.

5143 Figure 10.1 shows the chart that arises from parsing the sentence We eat sushi with
 5144 chopsticks using the grammar defined above.

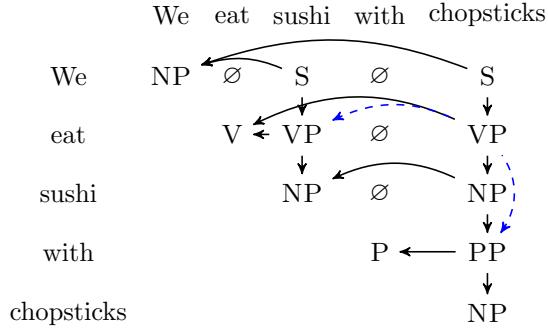


Figure 10.1: An example completed CKY chart. The solid and dashed lines show the back pointers resulting from the two different derivations of VP in position $t[1, 5]$.

5145 10.1.1 Recovering the parse tree

5146 As with the Viterbi algorithm, it is possible to identify a successful parse by storing and
 5147 traversing an additional table of back-pointers. If we add an entry X to cell $t[i, j]$ by
 5148 using the production $X \rightarrow YZ$ and the split point k , then we store the back-pointer
 5149 $b[i, j, X] = (Y, Z, k)$. Once the table is complete, we can recover a parse by tracing this
 5150 pointers, starting at $b[0, M, S]$, and stopping when they ground out at terminal productions.

5151 For ambiguous sentences, there will be multiple paths to reach $S \in t[0, M]$. For example,
 5152 in Figure 10.1, the goal state $S \in t[0, M]$ is reached through the state $VP \in t[1, 5]$, and
 5153 there are two different ways to generate this constituent: one with (eat sushi) and (with
 5154 chopsticks) as children, and another with (eat) and (sushi with chopsticks) as children.
 5155 The presence of multiple paths indicates that the input can be generated by the grammar
 5156 in more than one way. In Algorithm 13, one of these derivations is selected arbitrarily. As
 5157 discussed in § 10.3, weighted context-free grammars can select a single parse that maximizes
 5158 a scoring function.

5159 10.1.2 Non-binary productions

5160 The CKY algorithm assumes that all productions with non-terminals on the right-hand
 5161 side (RHS) are binary. But in real grammars, such as the one considered in chapter 9,
 5162 there will be productions with more than two elements on the right-hand side, and other
 5163 productions with only a single element.

- 5164 • Productions with more than two elements on the right-hand side can be binarized
 5165 by creating additional non-terminals, as described in § 9.2.1.2. For example, given
 5166 the production $VP \rightarrow V NP NP$ (for ditransitive verbs), we can convert to $VP \rightarrow$
 5167 $VP_{ditrans}/NP NP$, and then add the production $VP_{ditrans}/NP \rightarrow V NP$.

- What about unary productions like $VP \rightarrow V$? In practice, this is handled by making a second pass on each diagonal, in which each cell $t[i, j]$ is augmented with all possible unary productions capable of generating each item already in the cell — formally, $t[i, j]$ is extended to its unary closure. Suppose the example grammar in Table 10.1 were extended to include the production $VP \rightarrow V$, enabling sentences with intransitive verb phrases, like we eat. Then the cell $t[1, 2]$ — corresponding to the word eat — would first include the set $\{V\}$, and would be augmented to the set $\{V, VP\}$ during this second pass.

10.1.3 Complexity

For an input of length M and a grammar with R productions and N non-terminals, the space complexity of the CKY algorithm is $\mathcal{O}(M^2N)$: the number of cells in the chart is $\mathcal{O}(M^2)$, and each cell must hold $\mathcal{O}(N)$ elements. The time complexity is $\mathcal{O}(M^3R)$: each cell is computed by searching over $\mathcal{O}(M)$ split points, with R possible productions for each split point. Both the time and space complexity are considerably worse than the Viterbi algorithm, which is linear in the length of the input.

10.2 Ambiguity

Syntactic ambiguity is endemic to natural language. Here are a few broad categories:

- Attachment ambiguity: e.g., We eat sushi with chopsticks, I shot an elephant in my pajamas. In these examples, the prepositions (with, in) can attach to either the verb or the direct object.
- Modifier scope: e.g., southern food store, plastic cup holder. In these examples, the first word could be modifying the subsequent adjective, or the final noun.
- Particle versus preposition: e.g., The puppy tore up the staircase. Phrasal verbs like tore up often include particles which could also act as prepositions. This has structural implications: if up is a preposition, then up the staircase is a prepositional phrase; if up is a particle, then the staircase is the direct object to the verb.
- Complement structure: e.g., The students complained to the professor that they didn't understand. This is another form of attachment ambiguity, where the complement that they didn't understand could attach to the main verb (complained), or to the indirect object (the professor).
- Coordination scope: e.g., “I see,” said the blind man, as he picked up the hammer and saw. In this example, the lexical ambiguity for saw enables it to be coordinated either with the noun hammer or the verb picked up.

These forms of ambiguity can combine, so that seemingly simple headlines like Fed raises interest rates have dozens of possible analyses even in a minimal grammar. In a broad coverage grammar, typical sentences can have millions of parses. While careful grammar design can chip away at this ambiguity, a better strategy is combine broad coverage parsers with data driven strategies for identifying the correct analysis.

10.2.1 Parser evaluation

Before continuing to parsing algorithms that are able to handle ambiguity, we stop to consider how to measure parsing performance. Suppose we have a set of reference parses — the ground truth — and a set of system parses that we would like to score. A simple solution would be per-sentence accuracy: the parser is scored by the proportion of sentences on which the system and reference parses exactly match.² But as any good student knows, it is better to get partial credit, which we can assign to analyses that correctly match parts of the reference parse. The PARSEval metrics (Grishman et al., 1992) score each system parse via:

Precision: the fraction of constituents in the system parse that match a constituent in the reference parse.

Recall: the fraction of constituents in the reference parse that match a constituent in the system parse.

In labeled precision and recall, the system must also match the phrase type for each constituent; in unlabeled precision and recall, it is only required to match the constituent structure. As in chapter 4, the precision and recall can be combined into an *F*-measure, $F = \frac{2 \times P \times R}{P + R}$.

In Figure 10.2, suppose that the left tree is the system parse and the right tree is the reference parse. We have the following spans:

- $S \rightarrow w_{1:5}$ is true positive, because it appears in both trees.
- $VP \rightarrow w_{2:5}$ is true positive as well.
- $NP \rightarrow w_{3:5}$ is false positive, because it appears only in the system output.
- $PP \rightarrow w_{4:5}$ is true positive, because it appears in both trees.
- $VP \rightarrow w_{2:3}$ is false negative, because it appears only in the reference.

²Most parsing papers do not report results on this metric, but Finkel et al. (2008) find that a strong parser finds the exact correct parse on 35% of sentences of length ≤ 40 , and on 62% of parses of length ≤ 15 in the Penn Treebank.

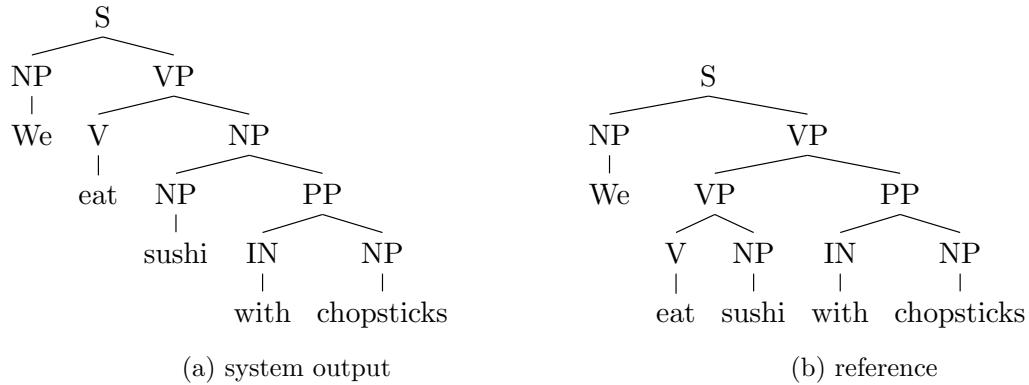


Figure 10.2: Two possible analyses from the grammar in Table 10.1

5230 The labeled and unlabeled precision of this parse is $\frac{3}{4} = 0.75$, and the recall is $\frac{3}{4} = 0.75$, for
 5231 an F-measure of 0.75. For an example in which precision and recall are not equal, suppose
 5232 the reference parse instead included the production $VP \rightarrow V NP PP$. In this parse, the
 5233 reference does not contain the constituent $w_{2:3}$, so the recall would be 1.³

5234 10.2.2 Local solutions

5235 Some ambiguity can be resolved locally. Consider the following examples,

- 5236 (10.1) We met the President on Monday.
5237 (10.2) We met the President of Mexico.

Each case ends with a preposition, which can be attached to the verb met or the noun phrase the president. This ambiguity can be resolved by using a labeled corpus to compare the likelihood of observing the preposition alongside each candidate attachment point,

$$p(\text{on} \mid \text{met}) \geq p(\text{on} \mid \text{President}) \quad [10.1]$$

$$p(\text{of} \mid \text{met}) \geq p(\text{of} \mid \text{President}). \quad [10.2]$$

5238 A comparison of these probabilities would successfully resolve this case (Hindle and Rooth,
 5239 1993). Other cases, such as the example ...eat sushi with chopsticks, require considering
 5240 the object of the preposition — consider the alternative ...eat sushi with soy sauce. With
 5241 sufficient labeled data, the problem of prepositional phrase attachment can be treated as
 5242 a classification task (Ratnaparkhi et al., 1994).

³While the grammar must be binarized before applying the CKY algorithm, evaluation is performed on the original parses. It is therefore necessary to “unbinarize” the output of a CKY-based parser, converting it back to the original grammar.

5243 However, there are inherent limitations to local solutions. While toy examples may
 5244 have just a few ambiguities to resolve, realistic sentences have thousands or millions of
 5245 possible parses. Furthermore, attachment decisions are interdependent, as shown in the
 5246 garden path example:

5247 (10.3) Cats scratch people with claws with knives.

5248 We may want to attach with claws to scratch, as would be correct in the shorter sentence in
 5249 cats scratch people with claws. But this leaves nowhere to attach with knives. The correct
 5250 interpretation can be identified only by considering the attachment decisions jointly. The
 5251 huge number of potential parses may seem to make exhaustive search impossible. But as
 5252 with sequence labeling, locality assumptions make it possible to search this space efficiently.

5253 10.3 Weighted Context-Free Grammars

5254 Let us define a derivation τ as a set of anchored productions,

$$\tau = \{X \rightarrow \alpha, (i, j, k)\}, \quad [10.3]$$

5255 with X corresponding to the left-hand side non-terminal and α corresponding to the
 5256 right-hand side. For grammars in Chomsky normal form, α is either a pair of non-terminals
 5257 or a terminal symbol. The indices i, j, k anchor the production in the input, with X deriving
 5258 the span $w_{i+1:j}$. For binary productions, $w_{i+1:k}$ indicates the span of the left child, and
 5259 $w_{k+1:j}$ indicates the span of the right child; for unary productions, k is ignored. For an
 5260 input w , the optimal parse is then,

$$\hat{\tau} = \underset{\tau \in \mathcal{T}(w)}{\operatorname{argmax}} \Psi(\tau), \quad [10.4]$$

5261 where $\mathcal{T}(w)$ is the set of derivations that yield the input w .

5262 The scoring function Ψ decomposes across anchored productions,

$$\Psi(\tau) = \sum_{(X \rightarrow \alpha, (i, j, k)) \in \tau} \psi(X \rightarrow \alpha, (i, j, k)). \quad [10.5]$$

5263 This is a locality assumption, akin to the assumption in Viterbi sequence labeling. In this
 5264 case, the assumption states that the overall score is a sum over scores of productions, which
 5265 are computed independently. In a weighted context-free grammar (WCFG), the score of
 5266 each anchored production $X \rightarrow (\alpha, i, j, k)$ is simply $\psi(X \rightarrow \alpha)$, ignoring the anchors
 5267 (i, j, k) . In other parsing models, the anchors can be used to access features of the input,
 5268 while still permitting efficient bottom-up parsing.

		$\psi(\cdot)$	$\exp \psi(\cdot)$
S	\rightarrow NP VP	0	1
NP	\rightarrow NP PP	-1	$\frac{1}{2}$
	\rightarrow we	-2	$\frac{1}{4}$
	\rightarrow sushi	-3	$\frac{1}{8}$
	\rightarrow chopsticks	-3	$\frac{1}{8}$
PP	\rightarrow In NP	0	1
In	\rightarrow with	0	1
VP	\rightarrow V NP	-1	$\frac{1}{2}$
	\rightarrow VP PP	-2	$\frac{1}{4}$
	\rightarrow Md V	-2	$\frac{1}{4}$
V	\rightarrow eat	0	1

Table 10.2: An example weighted context-free grammar (WCFG). The weights are chosen so that $\exp \psi(\cdot)$ sums to one over right-hand sides for each non-terminal; this is required by probabilistic context-free grammars, but not by WCFGs in general.

Example Consider the weighted grammar shown in Table 10.2, and the analysis in Figure 10.2b.

$$\begin{aligned} \Psi(\tau) &= \psi(S \rightarrow NP\ VP) + \psi(VP \rightarrow VP\ PP) + \psi(VP \rightarrow V\ NP) + \psi(PP \rightarrow In\ NP) \\ &\quad + \psi(NP \rightarrow We) + \psi(V \rightarrow eat) + \psi(NP \rightarrow sushi) + \psi(In \rightarrow with) + \psi(NP \rightarrow chopsticks) \end{aligned} \quad [10.6]$$

$$= 0 - 2 - 1 + 0 - 2 + 0 - 3 + 0 - 3 = -11. \quad [10.7]$$

5269 In the alternative parse in Figure 10.2a, the production $VP \rightarrow VP\ PP$ (with score -2) is
 5270 replaced with the production $NP \rightarrow NP\ PP$ (with score -1); all other productions are the
 5271 same. As a result, the score for this parse is -10.

5272 This example hints at a big problem with WCFG parsing on non-terminals such as NP,
 5273 VP, and PP: a WCFG will always prefer either VP or NP attachment, without regard to
 5274 what is being attached! This problem is addressed in § 10.5.

5275 10.3.1 Parsing with weighted context-free grammars

5276 The optimization problem in Equation 10.4 can be solved by modifying the CKY algorithm.
 5277 In the deterministic CKY algorithm, each cell $t[i, j]$ stored a set of non-terminals capable
 5278 of deriving the span $w_{i+1:j}$. We now augment the table so that the cell $t[i, j, X]$ is the score
 5279 of the best derivation of $w_{i+1:j}$ from non-terminal X . This score is computed recursively:
 5280 for the anchored binary production $(X \rightarrow Y\ Z, (i, j, k))$, we compute:

Algorithm 14 CKY algorithm for parsing a string $\mathbf{w} \in \Sigma^*$ 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( $\mathbf{w}, G = (N, \Sigma, R, S)$ )
    for all  $i, j, X$  do                                 $\triangleright$  Initialization
         $t[i, j, X] \leftarrow 0$ 
         $b[i, j, X] \leftarrow \emptyset$ 
    for  $m \in \{1, 2, \dots, M\}$  do
        for all  $X \in N$  do
             $t[m, m + 1, X] \leftarrow \psi(X \rightarrow w_m, (m, m + 1, m))$ 
    for  $\ell \in \{2, 3, \dots, M\}$  do
        for  $m \in \{0, 1, \dots, M - \ell\}$  do
            for  $k \in \{m + 1, m + 2, \dots, m + \ell - 1\}$  do
                 $t[m, m + \ell, X] \leftarrow \max_{k, Y, Z} \psi(X \rightarrow Y Z, (m, m + \ell, k)) + t[m, k, Y] + t[k, m + \ell, Z]$ 
                 $b[m, m + \ell, X] \leftarrow \operatorname{argmax}_{k, Y, Z} \psi(X \rightarrow Y Z, (m + \ell, k)) + t[m, k, Y] + t[k, m + \ell, Z]$ 
    return TraceBack( $S, 0, M, b$ )

```

- 5281 • the score of the anchored production, $\psi(X \rightarrow Y Z, (i, j, k))$;
- 5282 • the score of the best derivation of the left child, $t[i, k, Y]$;
- 5283 • the score of the best derivation of the right child, $t[k, j, Z]$.

5284 These scores are combined by addition. As in the unscored CKY algorithm, the table is
 5285 constructed by considering spans of increasing length, so the scores for spans $t[i, k, Y]$ and
 5286 $t[k, j, Z]$ are guaranteed to be available at the time we compute the score $t[i, j, X]$. The
 5287 value $t[0, M, S]$ is the score of the best derivation of \mathbf{w} from the grammar. Algorithm 14
 5288 formalizes this procedure.

5289 As in unweighted CKY, the parse is recovered from the table of back pointers b , where
 5290 each $b[i, j, X]$ stores the argmax split point k and production $X \rightarrow Y Z$ in the derivation of
 5291 $\mathbf{w}_{i+1:j}$ from X . The best parse can be obtained by tracing these pointers backwards from
 5292 $b[0, M, S]$, all the way to the terminal symbols. This is analogous to the computation of
 5293 the best sequence of labels in the Viterbi algorithm by tracing pointers backwards from the
 5294 end of the trellis. Note that we need only store back-pointers for the best path to $t[i, j, X]$;
 5295 this follows from the locality assumption that the global score for a parse is a combination
 5296 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:

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

```

$t[0, 1, \text{NP}] = -2$, $t[1, 2, \text{V}] = 0$, and so on. At the next diagonal, we compute the scores for spans of length 2: $t[1, 3, \text{VP}] = -1 + 0 - 3 = -4$, $t[3, 5, \text{PP}] = 0 + 0 - 3 = -3$, and so on. Things get interesting when we reach the cell $t[1, 5, \text{VP}]$, which contains the score for the derivation of the span $w_{2:5}$ from the non-terminal VP. This score is computed as a max over two alternatives,

$$t[1, 5, \text{VP}] = \max(\psi(\text{VP} \rightarrow \text{VP PP}, (1, 3, 5)) + t[1, 3, \text{VP}] + t[3, 5, \text{PP}], \\ \psi(\text{VP} \rightarrow \text{V NP}, (1, 2, 5)) + t[1, 2, \text{V}] + t[2, 5, \text{NP}]) \quad [10.8]$$

$$= \max(-2 - 4 - 3, -1 + 0 - 7) = -8. \quad [10.9]$$

5297 Since the second case is the argmax, we set the back-pointer $b[1, 5, \text{VP}] = (\text{V}, \text{NP}, 2)$,
5298 enabling the optimal derivation to be recovered.

5299 10.3.2 Probabilistic context-free grammars

5300 Probabilistic context-free grammars (PCFGs) are a special case of weighted context-free
5301 grammars that arises when the weights correspond to probabilities. Specifically, the weight
5302 $\psi(X \rightarrow \alpha, (i, j, k)) = \log p(\alpha | X)$, where the probability of the right-hand side α is
5303 conditioned on the non-terminal X . These probabilities must be normalized over all
5304 possible right-hand sides, so that $\sum_\alpha p(\alpha | X) = 1$, for all X . For a given parse τ ,
5305 the product of the probabilities of the productions is equal to $p(\tau)$, under the generative
5306 model $\tau \sim \text{DrawSubtree}(S)$, where the function DrawSubtree is defined in Algorithm 15.

5307 The conditional probability of a parse given a string is,

$$p(\tau | w) = \frac{p(\tau)}{\sum_{\tau' \in \mathcal{T}(w)} p(\tau')} = \frac{\exp \Psi(\tau)}{\sum_{\tau' \in \mathcal{T}(w)} \exp \Psi(\tau')}, \quad [10.10]$$

5308 where $\Psi(\tau) = \sum_{X \rightarrow \alpha, (i, j, k) \in \tau} \psi(X \rightarrow \alpha)$; the anchor is ignored. Because the probability
5309 is monotonic in the score $\Psi(\tau)$, the maximum likelihood parse can be identified by the
5310 CKY algorithm without modification. If a normalized probability $p(\tau | w)$ is required, the
5311 denominator of Equation 10.10 can be computed by the inside 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 in \mathcal{T} (we eat sushi with chopsticks), with scores $\Psi(\tau_1) = \log p(\tau_1) = -10$ and $\Psi(\tau_2) = \log p(\tau_2) = -11$. Therefore, the conditional probability $p(\tau_1 | \mathbf{w})$ is equal to,

$$p(\tau_1 | \mathbf{w}) = \frac{p(\tau_1)}{p(\tau_1) + p(\tau_2)} = \frac{\exp \Psi(\tau_1)}{\exp \Psi(\tau_1) + \exp \Psi(\tau_2)} = \frac{2^{-10}}{2^{-10} + 2^{-11}} = \frac{2}{3}. \quad [10.11]$$

5312 The inside recurrence The denominator of Equation 10.10 can be viewed as a language
 5313 model, summing over all valid derivations of the string \mathbf{w} ,

$$p(\mathbf{w}) = \sum_{\tau': \text{yield}(\tau')=\mathbf{w}} p(\tau'). \quad [10.12]$$

Just as the CKY algorithm makes it possible to maximize over all such analyses, with a few modifications it can also compute their sum. Each cell $t[i, j, X]$ must store the log probability of deriving $\mathbf{w}_{i+1:j}$ from non-terminal X . To compute this, we replace the maximization over split points k and productions $X \rightarrow Y Z$ with a “log-sum-exp” operation, which exponentiates the log probabilities of the production and the children, sums them in probability space, and then converts back to the log domain:

$$t[i, j, X] = \log \sum_{k, Y, Z} \exp (\psi(X \rightarrow Y Z) + t[i, k, Y] + t[k, j, Z]) \quad [10.13]$$

$$= \log \sum_{k, Y, Z} \exp (\log p(Y Z | X) + \log p(Y \rightarrow \mathbf{w}_{i+1:k}) + \log p(Z \rightarrow \mathbf{w}_{k+1:j})) \quad [10.14]$$

$$= \log \sum_{k, Y, Z} p(Y Z | X) \times p(Y \rightarrow \mathbf{w}_{i+1:k}) \times p(Z \rightarrow \mathbf{w}_{k+1:j}) \quad [10.15]$$

$$= \log \sum_{k, Y, Z} p(Y Z, \mathbf{w}_{i+1:k}, \mathbf{w}_{k+1:j} | X) \quad [10.16]$$

$$= \log p(X \rightarrow \mathbf{w}_{i+1:j}). \quad [10.17]$$

5314 This is called the inside recurrence, because it computes the probability of each subtree
 5315 as a combination of the probabilities of the smaller subtrees that are inside of it. The
 5316 name implies a corresponding outside recurrence, which computes the probability of a
 5317 non-terminal X spanning $\mathbf{w}_{i+1:j}$, joint with the outside context $(\mathbf{w}_{1:i}, \mathbf{w}_{j+1:M})$. This
 5318 recurrence is described in § 10.4.3. The inside and outside recurrences are analogous to the
 5319 forward and backward recurrences in probabilistic sequence labeling (see § 7.5.3.3). They
 5320 can be used to compute the marginal probabilities of individual anchored productions,
 5321 $p(X \rightarrow \alpha, (i, j, k) | \mathbf{w})$, summing over all possible derivations of \mathbf{w} .

5322 10.3.3 *Semiring weighted context-free grammars

The weighted and unweighted CKY algorithms can be unified with the inside recurrence using the same semiring notation described in § 7.7.3. The generalized recurrence is:

$$t[i, j, X] = \bigoplus_{k, Y, Z} \psi(X \rightarrow Y Z, (i, j, k)) \otimes t[i, k, Y] \otimes t[k, j, Z]. \quad [10.18]$$

5323 This recurrence subsumes all of the algorithms that we have encountered in this chapter.

5324 Unweighted CKY. When $\psi(X \rightarrow \alpha, (i, j, k))$ is a Boolean truth value $\{\top, \perp\}$, \otimes is logical
 5325 conjunction, and \bigoplus is logical disjunction, then we derive the CKY recurrence for
 5326 unweighted context-free grammars, discussed in § 10.1 and Algorithm 13.

5327 Weighted CKY. When $\psi(X \rightarrow \alpha, (i, j, k))$ is a scalar score, \otimes is addition, and \bigoplus is
 5328 maximization, then we derive the CKY recurrence for weighted context-free grammars,
 5329 discussed in § 10.3 and Algorithm 14. When $\psi(X \rightarrow \alpha, (i, j, k)) = \log p(\alpha | X)$,
 5330 this same setting derives the CKY recurrence for finding the maximum likelihood
 5331 derivation in a probabilistic context-free grammar.

5332 Inside recurrence. When $\psi(X \rightarrow \alpha, (i, j, k))$ is a log probability, \otimes is addition, and $\bigoplus =$
 5333 $\log \sum \exp$, then we derive the inside recurrence for probabilistic context-free grammars,
 5334 discussed in § 10.3.2. It is also possible to set $\psi(X \rightarrow \alpha, (i, j, k))$ directly equal to
 5335 the probability $p(\alpha | X)$. In this case, \otimes is multiplication, and \bigoplus is addition. While
 5336 this may seem more intuitive than working with log probabilities, there is the risk of
 5337 underflow on long inputs.

5338 Regardless of how the scores are combined, the key point is the locality assumption:
 5339 the score for a derivation is the combination of the independent scores for each anchored
 5340 production, and these scores do not depend on any other part of the derivation. For
 5341 example, if two non-terminals are siblings, the scores of productions from these non-terminals
 5342 are computed independently. This locality assumption is analogous to the first-order
 5343 Markov assumption in sequence labeling, where the score for transitions between tags
 5344 depends only on the previous tag and current tag, and not on the history. As with sequence
 5345 labeling, this assumption makes it possible to find the optimal parse efficiently; its linguistic
 5346 limitations are discussed in § 10.5.

5347 10.4 Learning weighted context-free grammars

5348 Like sequence labeling, context-free parsing is a form of structure prediction. As a result,
 5349 WCFGs can be learned using the same set of algorithms: generative probabilistic models,
 5350 structured perceptron, maximum conditional likelihood, and maximum margin learning.

5351 In all cases, learning requires a treebank, which is a dataset of sentences labeled with
 5352 context-free parses. Parsing research was catalyzed by the Penn Treebank (Marcus et al.,
 5353 1993), the first large-scale dataset of this type (see § 9.2.2). Phrase structure treebanks
 5354 exist for roughly two dozen other languages, with coverage mainly restricted to European
 5355 and East Asian languages, plus Arabic and Urdu.

5356 10.4.1 Probabilistic context-free grammars

Probabilistic context-free grammars are similar to hidden Markov models, in that they are generative models of text. In this case, the parameters of interest correspond to probabilities of productions, conditional on the left-hand side. As with hidden Markov models, these parameters can be estimated by relative frequency:

$$\psi(X \rightarrow \alpha) = \log p(X \rightarrow \alpha) \quad [10.19]$$

$$\hat{p}(X \rightarrow \alpha) = \frac{\text{count}(X \rightarrow \alpha)}{\text{count}(X)}. \quad [10.20]$$

5357 For example, the probability of the production $\text{NP} \rightarrow \text{Det Nn}$ is the corpus count of
 5358 this production, divided by the count of the non-terminal NP. This estimator applies to
 5359 terminal productions as well: the probability of $\text{Nn} \rightarrow \text{whale}$ is the count of how often
 5360 whale appears in the corpus as generated from an Nn tag, divided by the total count of the
 5361 Nn tag. Even with the largest treebanks — currently on the order of one million tokens
 5362 — it is difficult to accurately compute probabilities of even moderately rare events, such
 5363 as $\text{Nn} \rightarrow \text{whale}$. Therefore, smoothing is critical for making PCFGs effective.

5364 10.4.2 Feature-based parsing

5365 The scores for each production can be computed as an inner product of weights and features,

$$\psi(X \rightarrow \alpha) = \boldsymbol{\theta} \cdot \mathbf{f}(X, \alpha, (i, j, k), \mathbf{w}), \quad [10.21]$$

5366 where the feature vector $\mathbf{f}(X, \alpha)$ is a function of the left-hand side X , the right-hand side
 5367 α , the anchor indices (i, j, k) , and the input \mathbf{w} .

5368 The basic feature $\mathbf{f}(X, \alpha, (i, j, k)) = \{(X, \alpha)\}$ encodes only the identity of the production
 5369 itself, which is a discriminatively-trained model with the same expressiveness as a PCFG.
 5370 Features on anchored productions can include the words that border the span w_i, w_{j+1} , the
 5371 word at the split point w_{k+1} , the presence of a verb or noun in the left child span $w_{i+1:k}$,
 5372 and so on (Durrett and Klein, 2015). Scores on anchored productions can be incorporated
 5373 into CKY parsing without any modification to the algorithm, because it is still possible to
 5374 compute each element of the table $t[i, j, X]$ recursively from its immediate children.

5375 Other features can be obtained by grouping elements on either the left-hand or right-hand
 5376 side: for example it can be particularly beneficial to compute additional features by

5377 clustering terminal symbols, with features corresponding to groups of words with similar
 5378 syntactic properties. The clustering can be obtained from unlabeled datasets that are much
 5379 larger than any treebank, improving coverage. Such methods are described in chapter 14.

Feature-based parsing models can be estimated using the usual array of discriminative learning techniques. For example, a structure perceptron update can be computed as (Carreras et al., 2008),

$$\mathbf{f}(\tau, \mathbf{w}^{(i)}) = \sum_{(X \rightarrow \alpha, (i, j, k)) \in \tau} \mathbf{f}(X, \alpha, (i, j, k), \mathbf{w}^{(i)}) \quad [10.22]$$

$$\hat{\tau} = \operatorname{argmax}_{\tau \in \mathcal{T}(\mathbf{w})} \theta \cdot \mathbf{f}(\tau, \mathbf{w}^{(i)}) \quad [10.23]$$

$$\theta \leftarrow \mathbf{f}(\tau^{(i)}, \mathbf{w}^{(i)}) - \mathbf{f}(\hat{\tau}, \mathbf{w}^{(i)}). \quad [10.24]$$

5380 A margin-based objective can be optimized by selecting $\hat{\tau}$ through cost-augmented decoding
 5381 (\S 2.3.2), enforcing a margin of $\Delta(\hat{\tau}, \tau)$ between the hypothesis and the reference parse,
 5382 where Δ is a non-negative cost function, such as the Hamming loss (Stern et al., 2017).
 5383 It is also possible to train feature-based parsing models by conditional log-likelihood, as
 5384 described in the next section.

5385 10.4.3 *Conditional random field parsing

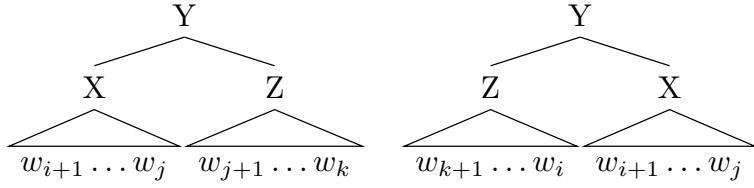
5386 The score of a derivation $\Psi(\tau)$ can be converted into a probability by normalizing over all
 5387 possible derivations,

$$p(\tau | \mathbf{w}) = \frac{\exp \Psi(\tau)}{\sum_{\tau' \in \mathcal{T}(\mathbf{w})} \exp \Psi(\tau')} \quad [10.25]$$

5388 Using this probability, a WCFG can be trained by maximizing the conditional log-likelihood
 5389 of a labeled corpus.

5390 Just as in logistic regression and the conditional random field over sequences, the
 5391 gradient of the conditional log-likelihood is the difference between the observed and expected
 5392 counts of each feature. The expectation $E_{\tau|\mathbf{w}}[\mathbf{f}(\tau, \mathbf{w}^{(i)}); \theta]$ requires summing over all
 5393 possible parses, and computing the marginal probabilities of anchored productions, $p(X \rightarrow$
 5394 $\alpha, (i, j, k) | \mathbf{w})$. In CRF sequence labeling, marginal probabilities over tag bigrams are
 5395 computed by the two-pass forward-backward algorithm (\S 7.5.3.3). The analogue for
 5396 context-free grammars is the inside-outside algorithm, in which marginal probabilities are
 5397 computed from terms generated by an upward and downward pass over the parsing chart:

- The upward pass is performed by the inside recurrence, which is described in \S 10.3.2.
 Each inside variable $\alpha(i, j, X)$ is the score of deriving $\mathbf{w}_{i+1:j}$ from the non-terminal
 X . In a PCFG, this corresponds to the log-probability $\log p(\mathbf{w}_{i+1:j} | X)$. This is

Figure 10.3: The two cases faced by the outside recurrence in the computation of $\beta(i, j, X)$

computed by the recurrence,

$$\alpha(i, j, X) \triangleq \log \sum_{(X \rightarrow Y \ Z)} \sum_{k=i+1}^j \exp(\psi(X \rightarrow Y \ Z, (i, j, k)) + \alpha(i, k, Y) + \alpha(k, j, Z)). \quad [10.26]$$

5398 The initial condition of this recurrence is $\alpha(m - 1, m, X) = \psi(X \rightarrow w_m)$. The
5399 denominator $\sum_{\tau \in \mathcal{T}(\mathbf{w})} \exp \Psi(\tau)$ is equal to $\exp \alpha(0, M, S)$.

- The downward pass is performed by the outside recurrence, which recursively populates the same table structure, starting at the root of the tree. Each outside variable $\beta(i, j, X)$ is the score of having a phrase of type X covering the span $(i + 1 : j)$, joint with the exterior context $\mathbf{w}_{1:i}$ and $\mathbf{w}_{j+1:M}$. In a PCFG, this corresponds to the log probability $\log p((X, i + 1, j), \mathbf{w}_{1:i}, \mathbf{w}_{j+1:M})$. Each outside variable is computed by the recurrence,

$$\exp \beta(i, j, X) \triangleq \sum_{(Y \rightarrow X \ Z)} \sum_{k=j+1}^M \exp [\psi(Y \rightarrow X \ Z, (i, k, j)) + \alpha(j, k, Z) + \beta(i, k, Y)] \quad [10.27]$$

$$+ \sum_{(Y \rightarrow Z \ X)} \sum_{k=0}^{i-1} \exp [\psi(Y \rightarrow Z \ X, (k, i, j)) + \alpha(k, i, Z) + \beta(k, j, Y)]. \quad [10.28]$$

5400 The first line of Equation 10.28 is the score under the condition that X is a left child
5401 of its parent, which spans $\mathbf{w}_{i+1:k}$, with $k > j$; the second line is the score under the
5402 condition that X is a right child of its parent Y , which spans $\mathbf{w}_{k+1:j}$, with $k < i$. The
5403 two cases are shown in Figure 10.3. In each case, we sum over all possible productions
5404 with X on the right-hand side. The parent Y is bounded on one side by either i or j ,
5405 depending on whether X is a left or right child of Y ; we must sum over all possible
5406 values for the other boundary. The initial conditions for the outside recurrence are
5407 $\beta(0, M, S) = 0$ and $\beta(0, M, X \neq S) = -\infty$.

The marginal probability of a non-terminal X over span $\mathbf{w}_{i+1:j}$ is written $p(X \rightsquigarrow \mathbf{w}_{i+1:j} | \mathbf{w})$, and can be computed from the inside and outside scores,

$$p(X \rightsquigarrow \mathbf{w}_{i+1:j} | \mathbf{w}) = \frac{p(X \rightsquigarrow \mathbf{w}_{i+1:j}, \mathbf{w})}{p(\mathbf{w})} \quad [10.29]$$

$$= \frac{p(\mathbf{w}_{i+1:j} | X) \times p(X, \mathbf{w}_{1:i}, \mathbf{x}_{j+1:M})}{p(\mathbf{w})} \quad [10.30]$$

$$= \frac{\exp(\alpha(i, j, X) + \beta(i, j, X))}{\exp \alpha(0, M, S)}. \quad [10.31]$$

Marginal probabilities of individual productions can be computed similarly (see exercise 2). These marginal probabilities can be used for training a conditional random field parser, and also for the task of unsupervised grammar induction, in which a PCFG is estimated from a dataset of unlabeled text (Lari and Young, 1990; Pereira and Schabes, 1992).

10.4.4 Neural context-free grammars

Recent work has applied neural representations to parsing, representing each span with a dense numerical vector (Socher et al., 2013; Durrett and Klein, 2015; Cross and Huang, 2016).⁴ For example, the anchor (i, j, k) and sentence \mathbf{w} can be associated with a fixed-length column vector,

$$\mathbf{v}_{(i,j,k)} = [\mathbf{u}_{w_{i-1}}; \mathbf{u}_{w_i}; \mathbf{u}_{w_{j-1}}; \mathbf{u}_{w_j}; \mathbf{u}_{w_{k-1}}; \mathbf{u}_{w_k}], \quad [10.32]$$

where \mathbf{u}_{w_i} is a word embedding associated with the word w_i . The vector $\mathbf{v}_{(i,j,k)}$ can then be passed through a feedforward neural network, and used to compute the score of the anchored production. For example, this score can be computed as a bilinear product (Durrett and Klein, 2015),

$$\tilde{\mathbf{v}}_{(i,j,k)} = \text{FeedForward}(\mathbf{v}_{(i,j,k)}) \quad [10.33]$$

$$\psi(X \rightarrow \alpha, (i, j, k)) = \tilde{\mathbf{v}}_{(i,j,k)}^\top \Theta \mathbf{f}(X \rightarrow \alpha), \quad [10.34]$$

where $\mathbf{f}(X \rightarrow \alpha)$ is a vector of discrete features of the production, and Θ is a parameter matrix. The matrix Θ and the parameters of the feedforward network can be learned by backpropagating from an objective such as the margin loss or the negative conditional log-likelihood.

10.5 Grammar refinement

The locality assumptions underlying CFG parsing depend on the granularity of the non-terminals. For the Penn Treebank non-terminals, there are several reasons to believe that these assumptions are too strong to enable accurate parsing (Johnson, 1998):

⁴Earlier work on neural constituent parsing used transition-based parsing algorithms (§ 10.6.2) rather than CKY-style chart parsing (Henderson, 2004; Titov and Henderson, 2007).

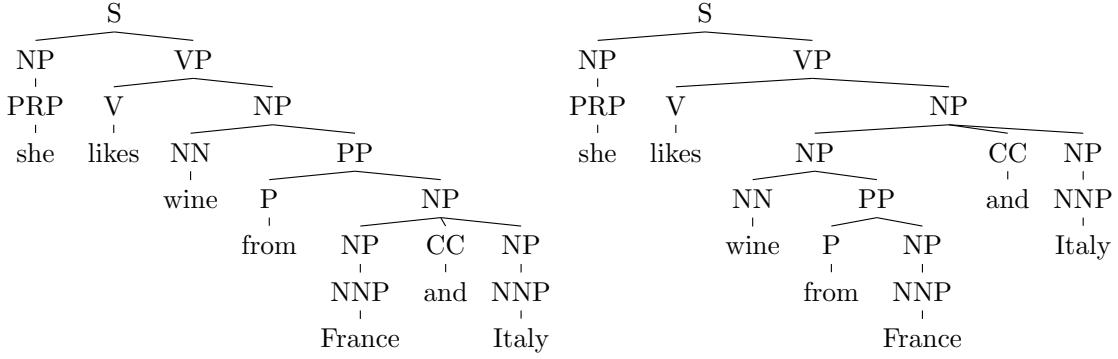


Figure 10.4: The left parse is preferable because of the conjunction of phrases headed by France and Italy, but these parses cannot be distinguished by a WCFG.

- The context-free assumption is too strict: for example, the probability of the production $NP \rightarrow NP\ PP$ is much higher (in the PTB) if the parent of the noun phrase is a verb phrase (indicating that the NP is a direct object) than if the parent is a sentence (indicating that the NP is the subject of the sentence).
- The Penn Treebank non-terminals are too coarse: there are many kinds of noun phrases and verb phrases, and accurate parsing sometimes requires knowing the difference. As we have already seen, when faced with prepositional phrase attachment ambiguity, a weighted CFG will either always choose NP attachment ($\psi(NP \rightarrow NP\ PP) > \psi(VP \rightarrow VP\ PP)$), or it will always choose VP attachment. To get more nuanced behavior, more fine-grained non-terminals are needed.
- More generally, accurate parsing requires some amount of semantics — understanding the meaning of the text to be parsed. Consider the example cats scratch people with claws: knowledge of about cats, claws, and scratching is necessary to correctly resolve the attachment ambiguity.

An extreme example is shown in Figure 10.4. The analysis on the left is preferred because of the conjunction of similar entities France and Italy. But given the non-terminals shown in the analyses, there is no way to differentiate these two parses, since they include exactly the same productions. What is needed seems to be more precise non-terminals. One possibility would be to rethink the linguistics behind the Penn Treebank, and ask the annotators to try again. But the original annotation effort took five years, and there is a little appetite for another annotation effort of this scope. Researchers have therefore turned to automated techniques.

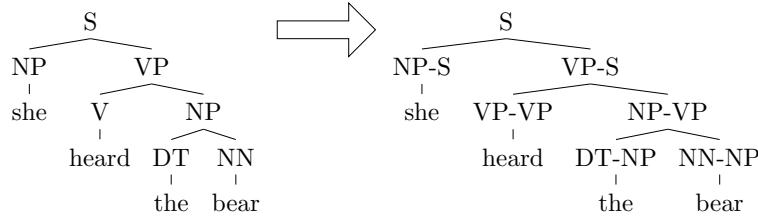


Figure 10.5: Parent annotation in a CFG derivation

5447 10.5.1 Parent annotations and other tree transformations

The key assumption underlying context-free parsing is that productions depend only on the identity of the non-terminal on the left-hand side, and not on its ancestors or neighbors. The validity of this assumption is an empirical question, and it depends on the non-terminals themselves: ideally, every noun phrase (and verb phrase, etc) would be distributionally identical, so the assumption would hold. But in the Penn Treebank, the observed probability of productions often depends on the parent of the left-hand side. For example, noun phrases are more likely to be modified by prepositional phrases when they are in the object position (e.g., they amused the students from Georgia) than in the subject position (e.g., the students from Georgia amused them). This means that the $\text{NP} \rightarrow \text{NP PP}$ production is more likely if the entire constituent is the child of a VP than if it is the child of S. The observed statistics are (Johnson, 1998):

$$\Pr(\text{NP} \rightarrow \text{NP PP}) = 11\% \quad [10.35]$$

$$\Pr(\text{NP under S} \rightarrow \text{NP PP}) = 9\% \quad [10.36]$$

$$\Pr(\text{NP under VP} \rightarrow \text{NP PP}) = 23\% \quad [10.37]$$

5448 This phenomenon can be captured by parent annotation (Johnson, 1998), in which each
 5449 non-terminal is augmented with the identity of its parent, as shown in Figure 10.5). This is
 5450 sometimes called vertical Markovization, since a Markov dependency is introduced between
 5451 each node and its parent (Klein and Manning, 2003). It is analogous to moving from a
 5452 bigram to a trigram context in a hidden Markov model. In principle, parent annotation
 5453 squares the size of the set of non-terminals, which could make parsing considerably less
 5454 efficient. But in practice, the increase in the number of non-terminals that actually appear
 5455 in the data is relatively modest (Johnson, 1998).

5456 Parent annotation weakens the WCFG locality assumptions. This improves accuracy
 5457 by enabling the parser to make more fine-grained distinctions, which better capture real
 5458 linguistic phenomena. However, each production is more rare, and so careful smoothing or
 5459 regularization is required to control the variance over production scores.

5460 10.5.2 Lexicalized context-free grammars

5461 The examples in § 10.2.2 demonstrate the importance of individual words in resolving
 5462 parsing ambiguity: the preposition on is more likely to attach to met, while the preposition
 5463 of is more likely to attach to President. But of all word pairs, which are relevant to
 5464 attachment decisions? Consider the following variants on the original examples:

5465 (10.4) We met the President of Mexico.

5466 (10.5) We met the first female President of Mexico.

5467 (10.6) They had supposedly met the President on Monday.

5468 The underlined words are the head words of their respective phrases: met heads the verb
 5469 phrase, and President heads the direct object noun phrase. These heads provide useful
 5470 semantic information. But they break the context-free assumption, which states that the
 5471 score for a production depends only on the parent and its immediate children, and not the
 5472 substructure under each child.

The incorporation of head words into context-free parsing is known as lexicalization,
 and is implemented in rules of the form,

$$\text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(of) \quad [10.38]$$

$$\text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(on). \quad [10.39]$$

5473 Lexicalization was a major step towards accurate PCFG parsing. It requires solving three
 5474 problems: identifying the heads of all constituents in a treebank; parsing efficiently while
 5475 keeping track of the heads; and estimating the scores for lexicalized productions.

5476 10.5.2.1 Identifying head words

5477 The head of a constituent is the word that is the most useful for determining how that
 5478 constituent is integrated into the rest of the sentence.⁵ The head word of a constituent
 5479 is determined recursively: for any non-terminal production, the head of the left-hand side
 5480 must be the head of one of the children. The head is typically selected according to a set
 5481 of deterministic rules, sometimes called head percolation rules. In many cases, these rules
 5482 are straightforward: the head of a noun phrase in a $\text{NP} \rightarrow \text{Det Nn}$ production is the head
 5483 of the noun; the head of a sentence in a $\text{S} \rightarrow \text{NP VP}$ production is the head of the verb
 5484 phrase.

5485 Table 10.3 shows a fragment of the head percolation rules used in many English parsing
 5486 systems. The meaning of the first rule is that to find the head of an S constituent, first
 5487 look for the rightmost VP child; if you don't find one, then look for the rightmost SBAR

⁵This is a pragmatic definition, befitting our goal of using head words to improve parsing; for a more formal definition, see (Bender, 2013, chapter 7).

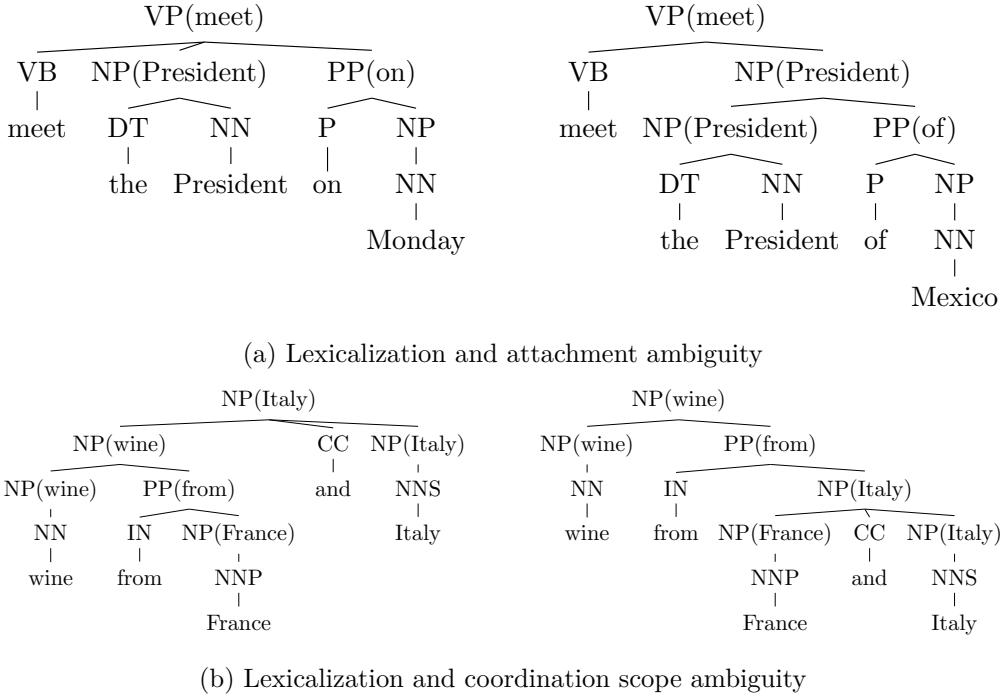


Figure 10.6: Examples of lexicalization

5488 child, and so on down the list. Verb phrases are headed by left verbs (the head of can
 5489 plan on walking is planned, since the modal verb can is tagged Md); noun phrases are
 5490 headed by the rightmost noun-like non-terminal (so the head of the red cat is cat),⁶ and
 5491 prepositional phrases are headed by the preposition (the head of at Georgia Tech is at).
 5492 Some of these rules are somewhat arbitrary — there's no particular reason why the head of
 5493 cats and dogs should be dogs — but the point here is just to get some lexical information
 5494 that can support parsing, not to make deep claims about syntax. Figure 10.6 shows the
 5495 application of these rules to two of the running examples.

5496 10.5.2.2 Parsing lexicalized context-free grammars

5497 A naïve application of lexicalization would simply increase the set of non-terminals by
 5498 taking the cross-product with the set of terminal symbols, so that the non-terminals now

⁶The noun phrase non-terminal is sometimes treated as a special case. Collins (1997) uses a heuristic that looks for the rightmost child which is a noun-like part-of-speech (e.g., Nn, Nnp), a possessive marker, or a superlative adjective (e.g., the greatest). If no such child is found, the heuristic then looks for the leftmost NP. If there is no child with tag NP, the heuristic then applies another priority list, this time from right to left.

Non-terminal	Direction	Priority
S	right	VP SBAR ADJP UCP NP
VP	left	VBD VBN MD VBZ TO VB VP VBG VBP ADJP NP
NP	right	N* EX \$ CD QP PRP ...
PP	left	IN TO FW

Table 10.3: A fragment of head percolation rules for English, from <http://www.cs.columbia.edu/~mcollins/papers/heads>

5499 include symbols like NP(President) and VP(meet). Under this approach, the CKY parsing
 5500 algorithm could be applied directly to the lexicalized production rules. However, the
 5501 complexity would be cubic in the size of the vocabulary of terminal symbols, which would
 5502 clearly be intractable.

Another approach is to augment the CKY table with an additional index, keeping track of the head of each constituent. The cell $t[i, j, h, X]$ stores the score of the best derivation in which non-terminal X spans $w_{i+1:j}$ with head word h , where $i < h \leq j$. To compute such a table recursively, we must consider the possibility that each phrase gets its head from either its left or right child. The scores of the best derivations in which the head comes from the left and right child are denoted t_ℓ and t_r respectively, leading to the following recurrence:

$$t_\ell[i, j, h, X] = \max_{(X \rightarrow YZ)} \max_{k > h} \max_{k < h' \leq j} t[i, k, h, Y] + t[k, j, h', Z] + \psi(X(h) \rightarrow Y(h)Z(h')) \quad [10.40]$$

$$t_r[i, j, h, X] = \max_{(X \rightarrow YZ)} \max_{k < h} \max_{i < h' \leq k} t[i, k, h', Y] + t[k, j, h, Z] + (\psi(X(h) \rightarrow Y(h')Z(h))) \quad [10.41]$$

$$t[i, j, h, X] = \max(t_\ell[i, j, h, X], t_r[i, j, h, X]). \quad [10.42]$$

5503 To compute t_ℓ , we maximize over all split points $k > h$, since the head word must be in
 5504 the left child. We then maximize again over possible head words h' for the right child. An
 5505 analogous computation is performed for t_r . The size of the table is now $\mathcal{O}(M^3N)$, where
 5506 M is the length of the input and N is the number of non-terminals. Furthermore, each
 5507 cell is computed by performing $\mathcal{O}(M^2)$ operations, since we maximize over both the split
 5508 point k and the head h' . The time complexity of the algorithm is therefore $\mathcal{O}(RM^5N)$,
 5509 where R is the number of rules in the grammar. Fortunately, more efficient solutions are
 5510 possible. In general, the complexity of parsing can be reduced to $\mathcal{O}(M^4)$ in the length of
 5511 the input; for a broad class of lexicalized CFGs, the complexity can be made cubic in the
 5512 length of the input, just as in unlexicalized CFGs (Eisner, 2000).

5513 10.5.2.3 Estimating lexicalized context-free grammars

5514 The final problem for lexicalized parsing is how to estimate weights for lexicalized productions
 5515 $X(i) \rightarrow Y(j) Z(k)$. These productions are said to be bilexical, because they involve
 5516 scores over pairs of words: in the example meet the President of Mexico, we hope to
 5517 choose the correct attachment point by modeling the bilexical affinities of (meet, of) and
 5518 (President, of). The number of such word pairs is quadratic in the size of the vocabulary,
 5519 making it difficult to estimate the weights of lexicalized production rules directly from
 5520 data. This is especially true for probabilistic context-free grammars, in which the weights
 5521 are obtained from smoothed relative frequency. In a treebank with a million tokens, a
 5522 vanishingly small fraction of the possible lexicalized productions will be observed more than
 5523 once.⁷ The Charniak (1997) and Collins (1997) parsers therefore focus on approximating
 5524 the probabilities of lexicalized productions, using various smoothing techniques and independence
 5525 assumptions.

In discriminatively-trained weighted context-free grammars, the scores for each production can be computed from a set of features, which can be made progressively more fine-grained (Finkel et al., 2008). For example, the score of the lexicalized production $NP(\text{President}) \rightarrow NP(\text{President}) PP(\text{of})$ can be computed from the following features:

$$\begin{aligned} f(NP(\text{President}) \rightarrow NP(\text{President}) PP(\text{of})) = & \{NP(*) \rightarrow NP(*) PP(*), \\ & NP(\text{President}) \rightarrow NP(\text{President}) PP(*), \\ & NP(*) \rightarrow NP(*) PP(\text{of}), \\ & NP(\text{President}) \rightarrow NP(\text{President}) PP(\text{of})\} \end{aligned}$$

5526 The first feature scores the unlexicalized production $NP \rightarrow NP PP$; the next two features
 5527 lexicalize only one element of the production, thereby scoring the appropriateness of NP
 5528 attachment for the individual words President and of ; the final feature scores the specific
 5529 bilexical affinity of President and of . For bilexical pairs that are encountered frequently in
 5530 the treebank, this bilexical feature can play an important role in parsing; for pairs that are
 5531 absent or rare, regularization will drive its weight to zero, forcing the parser to rely on the
 5532 more coarse-grained features.

5533 In chapter 14, we will encounter techniques for clustering words based on their distributional
 5534 properties — the contexts in which they appear. Such a clustering would group rare
 5535 and common words, such as whale , shark , beluga , Leviathan . Word clusters can be used
 5536 as features in discriminative lexicalized parsing, striking a middle ground between full
 5537 lexicalization and non-terminals (Finkel et al., 2008). In this way, labeled examples
 5538 containing relatively common words like whale can help to improve parsing for rare words
 5539 like beluga , as long as those two words are clustered together.

⁷The real situation is even more difficult, because non-binary context-free grammars can involve trilexical or higher-order dependencies, between the head of the constituent and multiple of its children (Carreras et al., 2008).

5540 10.5.3 *Refinement grammars

5541 Lexicalization improves on context-free parsing by adding detailed information in the form
 5542 of lexical heads. However, estimating the scores of lexicalized productions is difficult. Klein
 5543 and Manning (2003) argue that the right level of linguistic detail is somewhere between
 5544 treebank categories and individual words. Some parts-of-speech and non-terminals are
 5545 truly substitutable: for example, cat/N and dog/N. But others are not: for example,
 5546 the preposition of exclusively attaches to nouns, while the preposition as is more likely to
 5547 modify verb phrases. Klein and Manning (2003) obtained a 2% improvement in F -measure
 5548 on a parent-annotated PCFG parser by making a single change: splitting the preposition
 5549 category into six subtypes. They propose a series of linguistically-motivated refinements
 5550 to the Penn Treebank annotations, which in total yielded a 40% error reduction.

5551 Non-terminal refinement process can be automated by treating the refined categories as
 5552 latent variables. For example, we might split the noun phrase non-terminal into NP1, NP2, NP3, ...,
 5553 without defining in advance what each refined non-terminal corresponds to. This can
 5554 be treated as partially supervised learning, similar to the multi-component document
 5555 classification model described in § 5.2.3. A latent variable PCFG can be estimated by
 5556 expectation-maximization (Matsuzaki et al., 2005):

- 5557 • In the E-step, estimate a marginal distribution q over the refinement type of each
 5558 non-terminal in each derivation. These marginals are constrained by the original
 5559 annotation: an NP can be reannotated as NP4, but not as VP3. Marginal probabilities
 5560 over refined productions can be computed from the inside-outside algorithm, as
 5561 described in § 10.4.3, where the E-step enforces the constraints imposed by the
 5562 original annotations.
- 5563 • In the M-step, recompute the parameters of the grammar, by summing over the
 5564 probabilities of anchored productions that were computed in the E-step:

$$E[\text{count}(X \rightarrow Y Z)] = \sum_{i=0}^M \sum_{j=i}^M \sum_{k=i}^j p(X \rightarrow Y Z, (i, j, k) | \mathbf{w}). \quad [10.43]$$

5565 As usual, this process can be iterated to convergence. To determine the number of
 5566 refinement types for each tag, Petrov et al. (2006) apply a split-merge heuristic; Liang
 5567 et al. (2007) and Finkel et al. (2007) apply Bayesian nonparametrics (Cohen, 2016).

5568 Some examples of refined non-terminals are shown in Table 10.4. The proper nouns
 5569 differentiate months, first names, middle initials, last names, first names of places, and
 5570 second names of places; each of these will tend to appear in different parts of grammatical
 5571 productions. The personal pronouns differentiate grammatical role, with PRP-0 appearing
 5572 in subject position at the beginning of the sentence (note the capitalization), PRP-1
 5573 appearing in subject position but not at the beginning of the sentence, and PRP-2 appearing
 5574 in object position.

Proper nouns			
NNP-14	Oct.	Nov.	Sept.
NNP-12	John	Robert	James
NNP-2	J.	E.	L.
NNP-1	Bush	Noriega	Peters
NNP-15	New	San	Wall
NNP-3	York	Francisco	Street
Personal Pronouns			
PRP-0	It	He	I
PRP-1	it	he	they
PRP-2	it	them	him

Table 10.4: Examples of automatically refined non-terminals and some of the words that they generate (Petrov et al., 2006).

5575 10.6 Beyond context-free parsing

5576 In the context-free setting, the score for a parse is a combination of the scores of individual
 5577 productions. As we have seen, these models can be improved by using finer-grained
 5578 non-terminals, via parent-annotation, lexicalization, and automated refinement. However,
 5579 the inherent limitations to the expressiveness of context-free parsing motivate the consideration
 5580 of other search strategies. These strategies abandon the optimality guaranteed by bottom-up
 5581 parsing, in exchange for the freedom to consider arbitrary properties of the proposed parses.

5582 10.6.1 Reranking

5583 A simple way to relax the restrictions of context-free parsing is to perform a two-stage
 5584 process, in which a context-free parser generates a k -best list of candidates, and a reranker
 5585 then selects the best parse from this list (Charniak and Johnson, 2005; Collins and Koo,
 5586 2005). The reranker can be trained from an objective that is similar to multi-class classification:
 5587 the goal is to learn weights that assign a high score to the reference parse, or to the parse
 5588 on the k -best list that has the lowest error. In either case, the reranker need only evaluate
 5589 the K best parses, and so no context-free assumptions are necessary. This opens the door
 5590 to more expressive scoring functions:

- 5591 • It is possible to incorporate arbitrary non-local features, such as the structural
 5592 parallelism and right-branching orientation of the parse (Charniak and Johnson,
 5593 2005).
- 5594 • Reranking enables the use of recursive neural networks, in which each constituent
 5595 span $w_{i+1:j}$ receives a vector $u_{i,j}$ which is computed from the vector representations of

5596 its children, using a composition function that is linked to the production rule (Socher
 5597 et al., 2013), e.g.,

$$\mathbf{u}_{i,j} = f \left(\Theta_{X \rightarrow Y} Z \begin{bmatrix} \mathbf{u}_{i,k} \\ \mathbf{u}_{k,j} \end{bmatrix} \right) \quad [10.44]$$

5598 The overall score of the parse can then be computed from the final vector, $\Psi(\tau) =$
 5599 $\theta \mathbf{u}_{0,M}$.

5600 Reranking can yield substantial improvements in accuracy. The main limitation is that it
 5601 can only find the best parse among the K -best offered by the generator, so it is inherently
 5602 limited by the ability of the bottom-up parser to find high-quality candidates.

5603 10.6.2 Transition-based parsing

5604 Structure prediction can be viewed as a form of search. An alternative to bottom-up
 5605 parsing is to read the input from left-to-right, gradually building up a parse structure
 5606 through a series of transitions. Transition-based parsing is described in more detail in the
 5607 next chapter, in the context of dependency parsing. However, it can also be applied to
 5608 CFG parsing, as briefly described here.

5609 For any context-free grammar, there is an equivalent pushdown automaton, a model of
 5610 computation that accepts exactly those strings that can be derived from the grammar.
 5611 This computational model consumes the input from left to right, while pushing and
 5612 popping elements on a stack. This architecture provides a natural transition-based parsing
 5613 framework for context-free grammars, known as shift-reduce parsing.

5614 Shift-reduce parsing is a type of transition-based parsing, in which the parser can take
 5615 the following actions:

- 5616 • shift the next terminal symbol onto the stack;
- 5617 • unary-reduce the top item on the stack, using a unary production rule in the grammar;
- 5618 • binary-reduce the top two items onto the stack, using a binary production rule in the
 5619 grammar.

5620 The set of available actions is constrained by the situation: the parser can only shift if
 5621 there are remaining terminal symbols in the input, and it can only reduce if an applicable
 5622 production rule exists in the grammar. If the parser arrives at a state where the input has
 5623 been completely consumed, and the stack contains only the element S, then the input is
 5624 accepted. If the parser arrives at a non-accepting state where there are no possible actions,
 5625 the input is rejected. A parse error occurs if there is some action sequence that would
 5626 accept an input, but the parser does not find it.

5627 Example Consider the input we eat sushi and the grammar in Table 10.1. The input can
 5628 be parsed through the following sequence of actions:

- 5629 1. Shift the first token we onto the stack.
- 5630 2. Reduce the top item on the stack to NP, using the production $NP \rightarrow we$.
- 5631 3. Shift the next token eat onto the stack, and reduce it to V with the production
 5632 $V \rightarrow eat$.
- 5633 4. Shift the final token sushi onto the stack, and reduce it to NP. The input has been
 5634 completely consumed, and the stack contains $[NP, V, NP]$.
- 5635 5. Reduce the top two items using the production $VP \rightarrow V NP$. The stack now contains
 5636 $[VP, NP]$.
- 5637 6. Reduce the top two items using the production $S \rightarrow NP VP$. The stack now contains
 5638 $[S]$. Since the input is empty, this is an accepting state.

5639 One thing to notice from this example is that the number of shift actions is equal to the
 5640 length of the input. The number of reduce actions is equal to the number of non-terminals
 5641 in the analysis, which grows linearly in the length of the input. Thus, the overall time
 5642 complexity of shift-reduce parsing is linear in the length of the input (assuming the
 5643 complexity of each individual classification decision is constant in the length of the input).
 5644 This is far better than the cubic time complexity required by CKY parsing.

5645 Transition-based parsing as inference In general, it is not possible to guarantee that a
 5646 transition-based parser will find the optimal parse, $\text{argmax}_{\tau} \Psi(\tau; \mathbf{w})$, even under the usual
 5647 CFG independence assumptions. We could assign a score to each anchored parsing action
 5648 in each context, with $\psi(a, c)$ indicating the score of performing action a in context c .
 5649 One might imagine that transition-based parsing could efficiently find the derivation that
 5650 maximizes the sum of such scores. But this too would require backtracking and searching
 5651 over an exponentially large number of possible action sequences: if a bad decision is made
 5652 at the beginning of the derivation, then it may be impossible to recover the optimal
 5653 action sequence without backtracking to that early mistake. This is known as a search
 5654 error. Transition-based parsers can incorporate arbitrary features, without the restrictive
 5655 independence assumptions required by chart parsing; search errors are the price that must
 5656 be paid for this flexibility.

5657 Learning transition-based parsing Transition-based parsing can be combined with machine
 5658 learning by training a classifier to select the correct action in each situation. This classifier
 5659 is free to choose any feature of the input, the state of the parser, and the parse history.
 5660 However, there is no optimality guarantee: the parser may choose a suboptimal parse, due
 5661 to a mistake at the beginning of the analysis. Nonetheless, some of the strongest CFG

5662 parsers are based on the shift-reduce architecture, rather than CKY. A recent generation
 5663 of models links shift-reduce parsing with recurrent neural networks, updating a hidden
 5664 state vector while consuming the input (e.g., Cross and Huang, 2016; Dyer et al., 2016).
 5665 Learning algorithms for transition-based parsing are discussed in more detail in § 11.3.

5666 Exercises

5667 1. Design a grammar that handles English subject-verb agreement. Specifically, your
 5668 grammar should handle the examples below correctly:

5669 (10.7) a. She sings.

5670 b. We sing.

5671 (10.8) a. *She sing.

5672 b. *We sings.

5673 2. Extend your grammar from the previous problem to include the auxiliary verb can,
 5674 so that the following cases are handled:

5675 (10.9) a. She can sing.

5676 b. We can sing.

5677 (10.10) a. *She can sings.

5678 b. *We can sings.

5679 3. French requires subjects and verbs to agree in person and number, and it requires
 5680 determiners and nouns to agree in gender and number. Verbs and their objects need
 5681 not agree. Assuming that French has two genders (feminine and masculine), three
 5682 persons (first [me], second [you], third [her]), and two numbers (singular and plural),
 5683 how many productions are required to extend the following simple grammar to handle
 5684 agreement?

5685	S	\rightarrow	NP VP
	VP	\rightarrow	V V NP V NP NP
	NP	\rightarrow	Det Nn

5686 4. Consider the grammar:

5687	S	\rightarrow	NP VP
	VP	\rightarrow	V NP
	NP	\rightarrow	Jj NP
	NP	\rightarrow	fish (the animal)
	V	\rightarrow	fish (the action of fishing)
	Jj	\rightarrow	fish (a modifier, as in fish sauce or fish stew)

5688 Apply the CKY algorithm and identify all possible parses for the sentence fish fish
 5689 fish fish.

5690 5. Choose one of the possible parses for the previous problem, and show how it can be
 5691 derived by a series of shift-reduce actions.

5692 6. To handle VP coordination, a grammar includes the production $VP \rightarrow VP\ Cc\ VP$.
 5693 To handle adverbs, it also includes the production $VP \rightarrow VP\ Adv$. Assume all verbs
 5694 are generated from a sequence of unary productions, e.g., $VP \rightarrow V \rightarrow eat$.

5695 a) Show how to binarize the production $VP \rightarrow VP\ Cc\ VP$.

5696 b) Use your binarized grammar to parse the sentence They eat and drink together,
 5697 treating together as an adverb.

5698 c) Prove that a weighted CFG cannot distinguish the two possible derivations of
 5699 this sentence. Your explanation should focus on the productions in the original,
 5700 non-binary grammar.

5701 d) Explain what condition must hold for a parent-annotated WCFG to prefer the
 5702 derivation in which together modifies the coordination eat and drink.

7. Consider the following PCFG:

$$p(X \rightarrow X\ X) = \frac{1}{2} \quad [10.45]$$

$$p(X \rightarrow Y) = \frac{1}{2} \quad [10.46]$$

$$p(Y \rightarrow \sigma) = \frac{1}{|\Sigma|}, \forall \sigma \in \Sigma \quad [10.47]$$

5703 a) Compute the probability $p(\hat{\tau})$ of the maximum probability parse for a string
 5704 $w \in \Sigma^M$.

5705 b) Compute the conditional probability $p(\hat{\tau} | w)$.

5706 8. Context-free grammars can be used to parse the internal structure of words. Using the
 5707 weighted CKY algorithm and the following weighted context-free grammar, identify
 5708 the best parse for the sequence of morphological segments in+flame+able.

	S	→	V	0
	S	→	N	0
	S	→	J	0
	V	→	VPref N	-1
	J	→	N JSuff	1
5709	J	→	V JSuff	0
	J	→	NegPref J	1
	VPref	→	in+	2
	NegPref	→	in+	1
	N	→	flame	0
	JSuff	→	+able	0

- 5710 9. Use the inside and outside scores to compute the marginal probability $p(X_{i:j} \rightarrow Y_{i:k-1} Z_{k:j} \mid \mathbf{w})$,
 5711 indicating that Y spans $\mathbf{w}_{i:k-1}$, Z spans $\mathbf{w}_{k:j}$, and X is the parent of Y and Z ,
 5712 spanning $\mathbf{w}_{i:j}$.
- 5713 10. Suppose that the potentials $\Psi(X \rightarrow \alpha)$ are log-probabilities, so that $\sum_\alpha \exp \Psi(X \rightarrow \alpha) = 1$
 5714 for all X . Verify that the semiring inside recurrence from Equation 10.26 generates
 5715 the log-probability $\log p(\mathbf{w}) = \log \sum_{\tau: \text{yield}(\tau)=\mathbf{w}} p(\tau)$.

5716 Chapter 11

5717 Dependency parsing

5718 The previous chapter discussed algorithms for analyzing sentences in terms of nested
5719 constituents, such as noun phrases and verb phrases. However, many of the key sources of
5720 ambiguity in phrase-structure analysis relate to questions of attachment: where to attach a
5721 prepositional phrase or complement clause, how to scope a coordinating conjunction, and
5722 so on. These attachment decisions can be represented with a more lightweight structure:
5723 a directed graph over the words in the sentence, known as a dependency parse. Syntactic
5724 annotation has shifted its focus to such dependency structures: at the time of this writing,
5725 the Universal Dependencies project offers more than 100 dependency treebanks for more
5726 than 60 languages.¹ This chapter will describe the linguistic ideas underlying dependency
5727 grammar, and then discuss exact and transition-based parsing algorithms. The chapter will
5728 also discuss recent research on learning to search in transition-based structure prediction.

5729 11.1 Dependency grammar

5730 While dependency grammar has a rich history of its own (Tesnière, 1966; Kübler et al.,
5731 2009), it can be motivated by extension from the lexicalized context-free grammars that
5732 we encountered in previous chapter (§ 10.5.2). Recall that lexicalization augments each
5733 non-terminal with a head word. The head of a constituent is identified recursively, using
5734 a set of head rules, as shown in Table 10.3. An example of a lexicalized context-free parse
5735 is shown in Figure 11.1a. In this sentence, the head of the S constituent is the main verb,
5736 scratch; this non-terminal then produces the noun phrase the cats, whose head word is
5737 cats, and from which we finally derive the word the. Thus, the word scratch occupies the
5738 central position for the sentence, with the word cats playing a supporting role. In turn, cats
5739 occupies the central position for the noun phrase, with the word the playing a supporting
5740 role.

¹universaldependencies.org

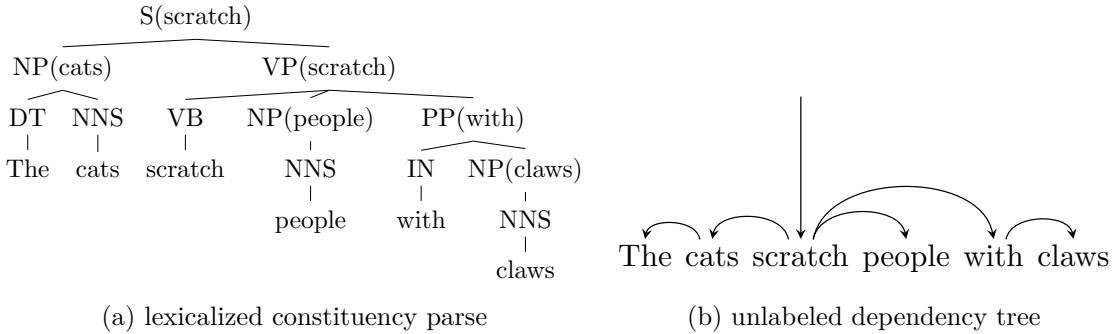


Figure 11.1: Dependency grammar is closely linked to lexicalized context free grammars: each lexical head has a dependency path to every other word in the constituent. (This example is based on the lexicalization rules from § 10.5.2, which make the preposition the head of a prepositional phrase. In the more contemporary Universal Dependencies annotations, the head of with claws would be claws, so there would be an edge scratch → claws.)

5741 The relationships between words in a sentence can be formalized in a directed graph,
 5742 based on the lexicalized phrase-structure parse: create an edge (i, j) iff word i is the head
 5743 of a phrase whose child is a phrase headed by word j . Thus, in our example, we would
 5744 have scratch → cats and cats → the. We would not have the edge scratch → the, because
 5745 although $S(\text{scratch})$ dominates $\text{Det}(\text{the})$ in the phrase-structure parse tree, it is not its
 5746 immediate parent. These edges describe syntactic dependencies, a bilexical relationship
 5747 between a head and a dependent, which is at the heart of dependency grammar.

5748 Continuing to build out this dependency graph, we will eventually reach every word in
 5749 the sentence, as shown in Figure 11.1b. In this graph — and in all graphs constructed in
 5750 this way — every word has exactly one incoming edge, except for the root word, which
 5751 is indicated by a special incoming arrow from above. Furthermore, the graph is weakly
 5752 connected: if the directed edges were replaced with undirected edges, there would be a
 5753 path between all pairs of nodes. From these properties, it can be shown that there are no
 5754 cycles in the graph (or else at least one node would have to have more than one incoming
 5755 edge), and therefore, the graph is a tree. Because the graph includes all vertices, it is a
 5756 spanning tree.

5757 11.1.1 Heads and dependents

5758 A dependency edge implies an asymmetric syntactic relationship between the head and
 5759 dependent words, sometimes called modifiers. For a pair like the cats or cats scratch, how
 5760 do we decide which is the head? Here are some possible criteria:

- The head sets the syntactic category of the construction: for example, nouns are the heads of noun phrases, and verbs are the heads of verb phrases.
- 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 making and the head, but this seems like 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.

11.1.2 Labeled dependencies

Edges may be labeled to indicate the nature of the syntactic relation that holds between the two elements. For example, in Figure 11.2, the label nsubj on the edge from like to Abigail indicates that the subtree headed by Abigail is the noun subject of the verb like; similarly, the label obj on the edge from like to kimchi indicates that the subtree headed by kimchi is the object.³ The negation not is treated as an adverbial modifier (advmod) on the noun jook.

²The latest and most specific guidelines are available at universaldependencies.org/guidelines.html

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.

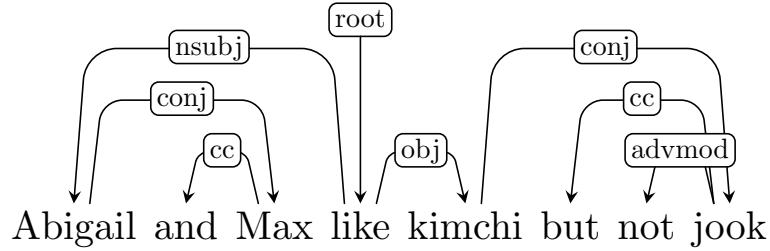


Figure 11.2: In the Universal Dependencies annotation system, the left-most item of a coordination is the head.

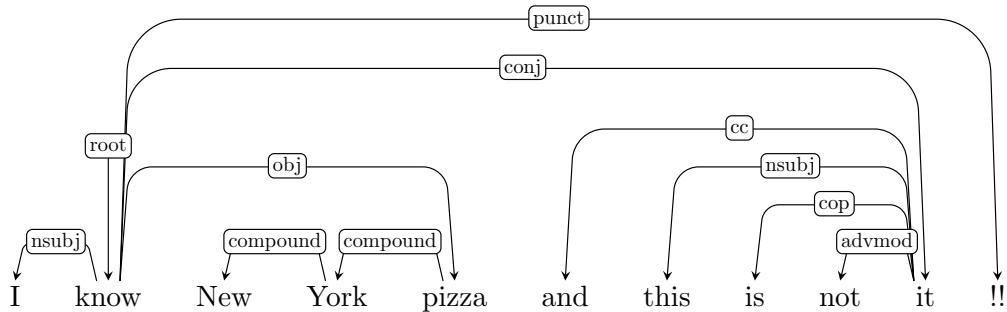


Figure 11.3: A labeled dependency parse from the English UD Treebank (reviews-361348-0006)

5795 A slightly more complex example is shown in Figure 11.3. The multiword expression
 5796 New York pizza is treated as a “flat” unit of text, with the elements linked by the compound
 5797 relation. The sentence includes two clauses that are conjoined in the same way that noun
 5798 phrases are conjoined in Figure 11.2. The second clause contains a copula verb (see § 8.1.1).
 5799 For such clauses, we treat the “object” of the verb as the root — in this case, it — and label
 5800 the verb as a dependent, with the cop relation. This example also shows how punctuations
 5801 are treated, with label punct.

5802 11.1.3 Dependency subtrees and constituents

5803 Dependency trees hide information that would be present in a CFG parse. Often what
 5804 is hidden is in fact irrelevant: for example, Figure 11.4 shows three different ways of
 5805 representing prepositional phrase adjuncts to the verb ate. Because there is apparently no
 5806 meaningful difference between these analyses, the Penn Treebank decides by convention
 5807 to use the two-level representation (see Johnson, 1998, for a discussion). As shown in
 5808 Figure 11.4d, these three cases all look the same in a dependency parse.

5809 But dependency grammar imposes its own set of annotation decisions, such as the

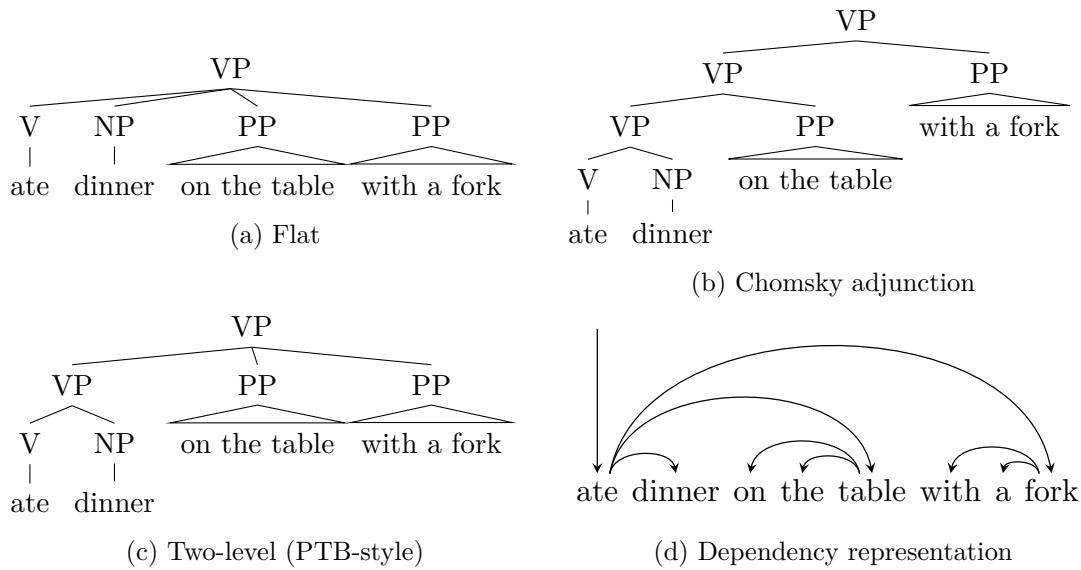


Figure 11.4: The three different CFG analyses of this verb phrase all correspond to a single dependency structure.

5810 identification of the head of a coordination (§ 11.1.1); without lexicalization, context-free
 5811 grammar does not require either element in a coordination to be privileged in this way.
 5812 Dependency parses can be disappointingly flat: for example, in the sentence Yesterday,
 5813 Abigail was reluctantly giving Max kimchi, the root giving is the head of every dependency!
 5814 The constituent parse arguably offers a more useful structural analysis for such cases.

5815 Projectivity Thus far, we have defined dependency trees as spanning trees over a graph
 5816 in which each word is a vertex. As we have seen, one way to construct such trees is
 5817 by connecting the heads in a lexicalized constituent parse. However, there are spanning
 5818 trees that cannot be constructed in this way. Syntactic constituents are contiguous spans.
 5819 In a spanning tree constructed from a lexicalized constituent parse, the head h of any
 5820 constituent that spans the nodes from i to j must have a path to every node in this span.
 5821 This property is known as projectivity, and projective dependency parses are a restricted
 5822 class of spanning trees. Informally, projectivity means that “crossing edges” are prohibited.
 5823 The formal definition follows:

5824 Definition 2 (Projectivity). An edge from i to j is projective iff all k between i and j are
 5825 descendants of i . A dependency parse is projective iff all its edges are projective.

5826 Figure 11.5 gives an example of a non-projective dependency graph in English. This
 5827 dependency graph does not correspond to any constituent parse. As shown in Table 11.1,

	% 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)

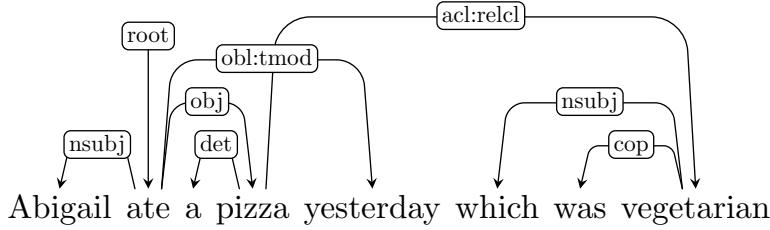


Figure 11.5: An example of a non-projective dependency parse. The “crossing edge” arises from the relative clause which was vegetarian and the oblique temporal modifier yesterday.

5828 non-projectivity is more common in languages such as Czech and German. Even though
 5829 relatively few dependencies are non-projective in these languages, many sentences have at
 5830 least one such dependency. As we will soon see, projectivity has important algorithmic
 5831 consequences.

5832 11.2 Graph-based dependency parsing

5833 Let $\mathbf{y} = \{i \xrightarrow{r} j\}$ represent a dependency graph, in which each edge is a relation r from
 5834 head word $i \in \{1, 2, \dots, M, \text{Root}\}$ to modifier $j \in \{1, 2, \dots, M\}$. The special node Root
 5835 indicates the root of the graph, and M is the length of the input $|\mathbf{w}|$. Given a scoring
 5836 function $\Psi(\mathbf{y}, \mathbf{w}; \theta)$, the optimal parse is,

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}(\mathbf{w})}{\operatorname{argmax}} \Psi(\mathbf{y}, \mathbf{w}; \theta), \quad [11.1]$$

5837 where $\mathcal{Y}(\mathbf{w})$ is the set of valid dependency parses on the input \mathbf{w} . As usual, the number
 5838 of possible labels $|\mathcal{Y}(\mathbf{w})|$ is exponential in the length of the input (Wu and Chao, 2004).
 5839 Algorithms that search over this space of possible graphs are known as graph-based dependency
 5840 parsers.

In sequence labeling and constituent parsing, it was possible to search efficiently over an exponential space by choosing a feature function that decomposes into a sum of local feature vectors. A similar approach is possible for dependency parsing, by requiring the

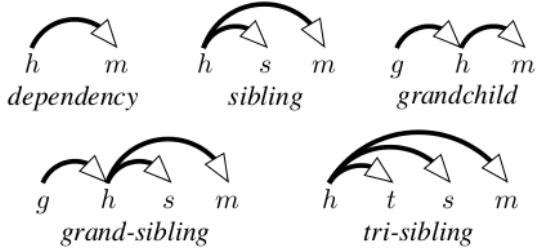


Figure 11.6: Feature templates for higher-order dependency parsing (Koo and Collins, 2010) [todo: permission]

scoring function to decompose across dependency arcs $i \rightarrow j$:

$$\Psi(\mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}). \quad [11.2]$$

5841 Dependency parsers that operate under this assumption are known as arc-factored, since
5842 the overall score is a product of scores over all arcs.

Higher-order dependency parsing The arc-factored decomposition can be relaxed to allow higher-order dependencies. In second-order dependency parsing, the scoring function may include grandparents and siblings, as shown by the templates in Figure 11.6. The scoring function is,

$$\begin{aligned} \Psi(\mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) = & \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi_{\text{parent}}(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) \\ & + \sum_{k \xrightarrow{r'} i \in \mathbf{y}} \psi_{\text{grandparent}}(i \xrightarrow{r} j, k, r', \mathbf{w}; \boldsymbol{\theta}) \end{aligned} \quad [11.3]$$

$$+ \sum_{\substack{i \xrightarrow{r'} s \in \mathbf{y} \\ s \neq j}} \psi_{\text{sibling}}(i \xrightarrow{r} j, s, r', \mathbf{w}; \boldsymbol{\theta}). \quad [11.4]$$

5843 The top line scores computes a scoring function that includes the grandparent k ; the bottom
5844 line computes a scoring function for each sibling s . For projective dependency graphs,
5845 there are efficient algorithms for second-order and third-order dependency parsing (Eisner,
5846 1996; McDonald and Pereira, 2006; Koo and Collins, 2010); for non-projective dependency
5847 graphs, second-order dependency parsing is NP-hard (McDonald and Pereira, 2006). The
5848 specific algorithms are discussed in the next section.

5849 11.2.1 Graph-based parsing algorithms

5850 The distinction between projective and non-projective dependency trees (§ 11.1.3) plays a
 5851 key role in the choice of algorithms. Because projective dependency trees are closely related
 5852 to (and can be derived from) lexicalized constituent trees, lexicalized parsing algorithms
 5853 can be applied directly. For the more general problem of parsing to arbitrary spanning
 5854 trees, a different class of algorithms is required. In both cases, arc-factored dependency
 5855 parsing relies on precomputing the scores $\psi(i \xrightarrow{r} j, \mathbf{w}; \theta)$ for each potential edge. There
 5856 are $\mathcal{O}(M^2R)$ such scores, where M is the length of the input and R is the number of
 5857 dependency relation types, and this is a lower bound on the time and space complexity of
 5858 any exact algorithm for arc-factored dependency parsing.

5859 11.2.1.1 Projective dependency parsing

5860 Any lexicalized constituency tree can be converted into a projective dependency tree by
 5861 creating arcs between the heads of constituents and their parents, so any algorithm for
 5862 lexicalized constituent parsing can be converted into an algorithm for projective dependency
 5863 parsing, by converting arc scores into scores for lexicalized productions. As noted in
 5864 § 10.5.2, there are cubic time algorithms for lexicalized constituent parsing, which are
 5865 extensions of the CKY algorithm. Therefore, arc-factored projective dependency parsing
 5866 can be performed in cubic time in the length of the input.

5867 Second-order projective dependency parsing can also be performed in cubic time, with
 5868 minimal modifications to the lexicalized parsing algorithm (Eisner, 1996). It is possible
 5869 to go even further, to third-order dependency parsing, in which the scoring function may
 5870 consider great-grandparents, grand-siblings, and “tri-siblings”, as shown in Figure 11.6.
 5871 Third-order dependency parsing can be performed in $\mathcal{O}(M^4)$ time, which can be made
 5872 practical through the use of pruning to eliminate unlikely edges (Koo and Collins, 2010).

5873 11.2.1.2 Non-projective dependency parsing

5874 In non-projective dependency parsing, the goal is to identify the highest-scoring spanning
 5875 tree over the words in the sentence. The arc-factored assumption ensures that the score
 5876 for each spanning tree will be computed as a sum over scores for the edges, which are
 5877 precomputed. Based on these scores, we build a weighted connected graph. Arc-factored
 5878 non-projective dependency parsing is then equivalent to finding the spanning tree that
 5879 achieves the maximum total score, $\Psi(\mathbf{y}, \mathbf{w}) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w})$. The Chu-Liu-Edmonds
 5880 algorithm (Chu and Liu, 1965; Edmonds, 1967) computes this maximum spanning tree
 5881 efficiently. It does this by first identifying the best incoming edge $i \xrightarrow{r} j$ for each vertex j .
 5882 If the resulting graph does not contain cycles, it is the maximum spanning tree. If there is
 5883 a cycle, it is collapsed into a super-vertex, whose incoming and outgoing edges are based

5884 on the edges to the vertices in the cycle. The algorithm is then applied recursively to the
 5885 resulting graph, and process repeats until a graph without cycles is obtained.

5886 The time complexity of identifying the best incoming edge for each vertex is $\mathcal{O}(M^2R)$,
 5887 where M is the length of the input and R is the number of relations; in the worst case, the
 5888 number of cycles is $\mathcal{O}(M)$. Therefore, the complexity of the Chu-Liu-Edmonds algorithm
 5889 is $\mathcal{O}(M^3R)$. This complexity can be reduced to $\mathcal{O}(M^2N)$ by storing the edge scores in a
 5890 Fibonacci heap (Gabow et al., 1986). For more detail on graph-based parsing algorithms,
 5891 see Eisner (1997) and Kübler et al. (2009).

5892 Higher-order non-projective dependency parsing Given the tractability of higher-order
 5893 projective dependency parsing, you may be surprised to learn that non-projective second-order
 5894 dependency parsing is NP-Hard. This can be proved by reduction from the vertex cover
 5895 problem (Neuhaus and Bröker, 1997). A heuristic solution is to do projective parsing first,
 5896 and then post-process the projective dependency parse to add non-projective edges (Nivre
 5897 and Nilsson, 2005). More recent work has applied techniques for approximate inference in
 5898 graphical models, including belief propagation (Smith and Eisner, 2008), integer linear
 5899 programming (Martins et al., 2009), variational inference (Martins et al., 2010), and
 5900 Markov Chain Monte Carlo (Zhang et al., 2014).

5901 11.2.2 Computing scores for dependency arcs

The arc-factored scoring function $\psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})$ can be defined in several ways:

$$\text{Linear} \quad \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \boldsymbol{\theta} \cdot \mathbf{f}(i \xrightarrow{r} j, \mathbf{w}) \quad [11.5]$$

$$\text{Neural} \quad \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \text{Feedforward}([\mathbf{u}_{w_i}; \mathbf{u}_{w_j}]; \boldsymbol{\theta}) \quad [11.6]$$

$$\text{Generative} \quad \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \log p(w_j, r | w_i). \quad [11.7]$$

5902 11.2.2.1 Linear feature-based arc scores

5903 Linear models for dependency parsing incorporate many of the same features used in
 5904 sequence labeling and discriminative constituent parsing. These include:

- 5905 • the length and direction of the arc;
- 5906 • the words w_i and w_j linked by the dependency relation;
- 5907 • the prefixes, suffixes, and parts-of-speech of these words;
- 5908 • the neighbors of the dependency arc, $w_{i-1}, w_{i+1}, w_{j-1}, w_{j+1}$;
- 5909 • the prefixes, suffixes, and part-of-speech of these neighbor words.

5910 Each of these features can be conjoined with the dependency edge label r . Note that
 5911 features in an arc-factored parser can refer to words other than w_i and w_j . The restriction
 5912 is that the features consider only a single arc.

Bilexical features (e.g., *sushi* → *chopsticks*) are powerful but rare, so it is useful to augment them with coarse-grained alternatives, by “backing off” to the part-of-speech or affix. For example, the following features are created by backing off to part-of-speech tags in an unlabeled dependency parser:

$$\begin{aligned} \mathbf{f}(3 \rightarrow 5, \text{we eat sushi with chopsticks}) = & \langle \text{sushi} \rightarrow \text{chopsticks}, \\ & \text{sushi} \rightarrow \text{Nns}, \\ & \text{Nn} \rightarrow \text{chopsticks}, \\ & \text{Nns} \rightarrow \text{Nn} \rangle. \end{aligned}$$

5913 Regularized discriminative learning algorithms can then trade off between features at
 5914 varying levels of detail. McDonald et al. (2005) take this approach as far as tetralexical
 5915 features (e.g., $(w_i, w_{i+1}, w_{j-1}, w_j)$). Such features help to avoid choosing arcs that are
 5916 unlikely due to the intervening words: for example, there is unlikely to be an edge between
 5917 two nouns if the intervening span contains a verb. A large list of first and second-order
 5918 features is provided by Bohnet (2010), who uses a hashing function to store these features
 5919 efficiently.

5920 11.2.2.2 Neural arc scores

Given vector representations \mathbf{x}_i for each word w_i in the input, a set of arc scores can be computed from a feedforward neural network:

$$\psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \text{FeedForward}([\mathbf{x}_i; \mathbf{x}_j]; \boldsymbol{\theta}_r), \quad [11.8]$$

where unique weights $\boldsymbol{\theta}_r$ are available for each arc type (Pei et al., 2015; Kiperwasser and Goldberg, 2016). Kiperwasser and Goldberg (2016) use a feedforward network with a single hidden layer,

$$\mathbf{z} = g(\boldsymbol{\Theta}_r[\mathbf{x}_i; \mathbf{x}_j] + b_r^{(z)}) \quad [11.9]$$

$$\psi(i \xrightarrow{r} j) = \boldsymbol{\beta}_r \mathbf{z} + b_r^{(y)}, \quad [11.10]$$

5921 where $\boldsymbol{\Theta}_r$ is a matrix, $\boldsymbol{\beta}_r$ is a vector, each b_r is a scalar, and the function g is an elementwise
 5922 \tanh activation function.

5923 The vector \mathbf{x}_i can be set equal to the word embedding, which may be pre-trained or
 5924 learned by backpropagation (Pei et al., 2015). Alternatively, contextual information can
 5925 be incorporated by applying a bidirectional recurrent neural network across the input, as
 5926 described in § 7.6. The RNN hidden states at each word can be used as inputs to the arc
 5927 scoring function (Kiperwasser and Goldberg, 2016).

5928 11.2.2.3 Probabilistic arc scores

If each arc score is equal to the log probability $\log p(w_j, r \mid w_i)$, then the sum of scores gives the log probability of the sentence and arc labels, by the chain rule. For example, consider the unlabeled parse of we eat sushi with rice,

$$\mathbf{y} = \{(root, 2), (2, 1), (2, 3), (3, 5), (5, 4)\} \quad [11.11]$$

$$\log p(\mathbf{w} \mid \mathbf{y}) = \sum_{(i \rightarrow j) \in \mathbf{y}} \log p(w_j \mid w_i) \quad [11.12]$$

$$\begin{aligned} &= \log p(\text{eat} \mid \text{root}) + \log p(\text{we} \mid \text{eat}) + \log p(\text{sushi} \mid \text{eat}) \\ &\quad + \log p(\text{rice} \mid \text{sushi}) + \log p(\text{with} \mid \text{rice}). \end{aligned} \quad [11.13]$$

5929 Probabilistic generative models are used in combination with expectation-maximization
5930 (chapter 5) for unsupervised dependency parsing (Klein and Manning, 2004).

5931 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,

$$\hat{\mathbf{y}} = \underset{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})}{\operatorname{argmax}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \mathbf{y}') \quad [11.14]$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \mathbf{f}(\mathbf{w}, \mathbf{y}) - \mathbf{f}(\mathbf{w}, \hat{\mathbf{y}}) \quad [11.15]$$

5932 In this case, the argmax requires a maximization over all dependency trees for the sentence,
5933 which can be computed using the algorithms described in § 11.2.1. We can apply all the
5934 usual tricks from § 2.2: weight averaging, a large margin objective, and regularization.
5935 McDonald et al. (2005) were the first to treat dependency parsing as a structure prediction
5936 problem, using MIRA, an online margin-based learning algorithm. Neural arc scores can be
5937 learned in the same way, backpropagating from a margin loss to updates on the feedforward
5938 network that computes the score for each edge.

A conditional random field for arc-factored dependency parsing is built on the probability model,

$$p(\mathbf{y} \mid \mathbf{w}) = \frac{\exp \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})}{\sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp \sum_{i \xrightarrow{r} j \in \mathbf{y}'} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})} \quad [11.16]$$

5939 Such a model is trained to minimize the negative log conditional-likelihood. Just as in CRF
5940 sequence models (§ 7.5.3) and the logistic regression classifier (§ 2.4), the gradients involve

5941 marginal probabilities $p(i \xrightarrow{r} j \mid \mathbf{w}; \theta)$, which in this case are probabilities over individual
 5942 dependencies. In arc-factored models, these probabilities can be computed in polynomial
 5943 time. For projective dependency trees, the marginal probabilities can be computed in cubic
 5944 time, using a variant of the inside-outside algorithm (Lari and Young, 1990). For
 5945 non-projective dependency parsing, marginals can also be computed in cubic time, using
 5946 the matrix-tree theorem (Koo et al., 2007; McDonald et al., 2007; Smith and Smith, 2007).
 5947 Details of these methods are described by Kübler et al. (2009).

5948 11.3 Transition-based dependency parsing

5949 Graph-based dependency parsing offers exact inference, meaning that it is possible to
 5950 recover the best-scoring parse for any given model. But this comes at a price: the scoring
 5951 function is required to decompose into local parts — in the case of non-projective parsing,
 5952 these parts are restricted to individual arcs. These limitations are felt more keenly in
 5953 dependency parsing than in sequence labeling, because second-order dependency features
 5954 are critical to correctly identify some types of attachments. For example, prepositional
 5955 phrase attachment depends on the attachment point, the object of the preposition, and
 5956 the preposition itself; arc-factored scores cannot account for all three of these features
 5957 simultaneously. Graph-based dependency parsing may also be criticized on the basis of
 5958 intuitions about human language processing: people read and listen to sentences sequentially,
 5959 incrementally building mental models of the sentence structure and meaning before getting
 5960 to the end (Jurafsky, 1996). This seems hard to reconcile with graph-based algorithms,
 5961 which perform bottom-up operations on the entire sentence, requiring the parser to keep
 5962 every word in memory. Finally, from a practical perspective, graph-based dependency
 5963 parsing is relatively slow, running in cubic time in the length of the input.

5964 Transition-based algorithms address all three of these objections. They work by moving
 5965 through the sentence sequentially, while performing actions that incrementally update a
 5966 stored representation of what has been read thus far. As with the shift-reduce parser
 5967 from § 10.6.2, this representation consists of a stack, onto which parsing substructures
 5968 can be pushed and popped. In shift-reduce, these substructures were constituents; in
 5969 the transition systems that follow, they will be projective dependency trees over partial
 5970 spans of the input.⁴ Parsing is complete when the input is consumed and there is only a
 5971 single structure on the stack. The sequence of actions that led to the parse is known as
 5972 the derivation. One problem with transition-based systems is that there may be multiple
 5973 derivations for a single parse structure — a phenomenon known as spurious ambiguity.

⁴Transition systems also exist for non-projective dependency parsing (e.g., Nivre, 2008).

5974 11.3.1 Transition systems for dependency parsing

5975 A transition system consists of a representation for describing configurations of the parser,
 5976 and a set of transition actions, which manipulate the configuration. There are two main
 5977 transition systems for dependency parsing: arc-standard, which is closely related to shift-reduce,
 5978 and arc-eager, which adds an additional action that can simplify derivations (Abney and
 5979 Johnson, 1991). In both cases, transitions are between configurations that are represented
 5980 as triples, $C = (\sigma, \beta, A)$, where σ is the stack, β is the input buffer, and A is the list of
 5981 arcs that have been created (Nivre, 2008). In the initial configuration,

$$C_{\text{initial}} = ([\text{Root}], \mathbf{w}, \emptyset), \quad [11.17]$$

5982 indicating that the stack contains only the special node Root, the entire input is on the
 5983 buffer, and the set of arcs is empty. An accepting configuration is,

$$C_{\text{accept}} = ([\text{Root}], \emptyset, A), \quad [11.18]$$

5984 where the stack contains only Root, the buffer is empty, and the arcs A define a spanning
 5985 tree over the input. The arc-standard and arc-eager systems define a set of transitions
 5986 between configurations, which are capable of transforming an initial configuration into
 5987 an accepting configuration. In both of these systems, the number of actions required to
 5988 parse an input grows linearly in the length of the input, making transition-based parsing
 5989 considerably more efficient than graph-based methods.

5990 11.3.1.1 Arc-standard

5991 The arc-standard transition system is closely related to shift-reduce, and to the LR algorithm
 5992 that is used to parse programming languages (Aho et al., 2006). It includes the following
 5993 classes of actions:

- 5994 • Shift: move the first item from the input buffer on to the top of the stack,

$$(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A), \quad [11.19]$$

5995 where we write $i|\beta$ to indicate that i is the leftmost item in the input buffer, and $\sigma|i$
 5996 to indicate the result of pushing i on to stack σ .

- 5997 • Arc-left: create a new left-facing arc of type r between the item on the top of the
 5998 stack and the first item in the input buffer. The head of this arc is j , which remains
 5999 at the front of the input buffer. The arc $j \xrightarrow{r} i$ is added to A . Formally,

$$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \oplus j \xrightarrow{r} i), \quad [11.20]$$

6000 where r is the label of the dependency arc, and \oplus concatenates the new arc $j \xrightarrow{r} i$ to
 6001 the list A .

σ	β	action	arc added to \mathcal{A}
1. [Root]	they like bagels with lox	Shift	
2. [Root, they]	like bagels with lox	Arc-Left	(they \leftarrow like)
3. [Root]	like bagels with lox	Shift	
4. [Root, like]	bagels with lox	Shift	
5. [Root, like, bagels]	with lox	Shift	
6. [Root, like, bagels, with]	lox	Arc-Left	(with \leftarrow lox)
7. [Root, like, bagels]	lox	Arc-Right	(bagels \rightarrow lox)
8. [Root, like]	bagels	Arc-Right	(like \rightarrow bagels)
9. [Root]	like	Arc-Right	(Root \rightarrow like)
10. [Root]	\emptyset	Done	

Table 11.2: Arc-standard derivation of the unlabeled dependency parse for the input they like bagels with lox.

- 6002 • Arc-right: creates a new right-facing arc of type r between the item on the top of
 6003 the stack and the first item in the input buffer. The head of this arc is i , which is
 6004 “popped” from the stack and pushed to the front of the input buffer. The arc $i \xrightarrow{r} j$
 6005 is added to A . Formally,

$$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, i|\beta, A \oplus i \xrightarrow{r} j), \quad [11.21]$$

6006 where again r is the label of the dependency arc.

6007 Each action has preconditions. The Shift action can be performed only when the buffer has
 6008 at least one element. The Arc-left action cannot be performed when the root node Root
 6009 is on top of the stack, since this node must be the root of the entire tree. The Arc-left
 6010 and Arc-right remove the modifier words from the stack (in the case of Arc-left) and from
 6011 the buffer (in the case of Arc-Right), so it is impossible for any word to have more than
 6012 one parent. Furthermore, the end state can only be reached when every word is removed
 6013 from the buffer and stack, so the set of arcs is guaranteed to constitute a spanning tree.
 6014 An example arc-standard derivation is shown in Table 11.2.

6015 11.3.1.2 Arc-eager dependency parsing

6016 In the arc-standard transition system, a word is completely removed from the parse once it
 6017 has been made the modifier in a dependency arc. At this time, any dependents of this word
 6018 must have already been identified. Right-branching structures are common in English (and
 6019 many other languages), with words often modified by units such as prepositional phrases
 6020 to their right. In the arc-standard system, this means that we must first shift all the
 6021 units of the input onto the stack, and then work backwards, creating a series of arcs, as

σ	β	action	arc added to \mathcal{A}
1. [Root]	they like bagels with lox	Shift	
2. [Root, they]	like bagels with lox	Arc-Left	(they \leftarrow like)
3. [Root]	like bagels with lox	Arc-Right	(Root \rightarrow like)
4. [Root, like]	bagels with lox	Arc-Right	(like \rightarrow bagels)
5. [Root, like, bagels]	with lox	Shift	
6. [Root, like, bagels, with]	lox	Arc-Left	(with \leftarrow lox)
7. [Root, like, bagels]	lox	Arc-Right	(bagels \rightarrow lox)
8. [Root, like, bagels, lox]	\emptyset	Reduce	
9. [Root, like, bagels]	\emptyset	Reduce	
10. [Root, like]	\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.

6022 occurs in Table 11.2. Note that the decision to shift bagels onto the stack guarantees that
 6023 the prepositional phrase with lox will attach to the noun phrase, and that this decision
 6024 must be made before the prepositional phrase is itself parsed. This has been argued to be
 6025 cognitively implausible (Abney and Johnson, 1991); from a computational perspective, it
 6026 means that a parser may need to look several steps ahead to make the correct decision.

6027 Arc-eager dependency parsing changes the arc-right action so that right dependents
 6028 can be attached before all of their dependents have been found. Rather than removing the
 6029 modifier from both the buffer and stack, the arc-right action pushes the modifier on to the
 6030 stack, on top of the head. Because the stack can now contain elements that already have
 6031 parents in the partial dependency graph, two additional changes are necessary:

- 6032 • A precondition is required to ensure that the arc-left action cannot be applied when
 6033 the top element on the stack already has a parent in A .
- 6034 • A new reduce action is introduced, which can remove elements from the stack if they
 6035 already have a parent in A :

$$(\sigma|i, \beta, A) \Rightarrow (\sigma, \beta, A). \quad [11.22]$$

6036 As a result of these changes, it is now possible to create the arc like \rightarrow bagels before parsing
 6037 the prepositional phrase with lox. Furthermore, this action does not imply a decision about
 6038 whether the prepositional phrase will attach to the noun or verb. Noun attachment is
 6039 chosen in the parse in Table 11.3, but verb attachment could be achieved by applying the
 6040 reduce action at step 5 or 7.

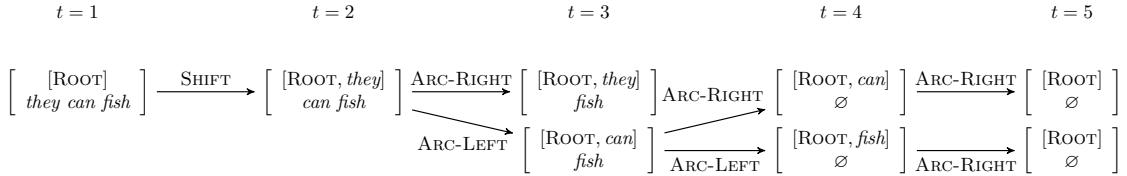


Figure 11.7: Beam search for unlabeled dependency parsing, with beam size $K = 2$. The arc lists for each configuration are not shown, but can be computed from the transitions.

6041 11.3.1.3 Projectivity

6042 The arc-standard and arc-eager transition systems are guaranteed to produce projective
 6043 dependency trees, because all arcs are between the word at the top of the stack and
 6044 the left-most edge of the buffer (Nivre, 2008). Non-projective transition systems can be
 6045 constructed by adding actions that create arcs with words that are second or third in
 6046 the stack (Attardi, 2006), or by adopting an alternative configuration structure, which
 6047 maintains a list of all words that do not yet have heads (Covington, 2001). In pseudo-projective
 6048 dependency parsing, a projective dependency parse is generated first, and then a set of
 6049 graph transformation techniques are applied, producing non-projective edges (Nivre and
 6050 Nilsson, 2005).

6051 11.3.1.4 Beam search

6052 In “greedy” transition-based parsing, the parser tries to make the best decision at each
 6053 configuration. This can lead to search errors, when an early decision locks the parser into
 6054 a poor derivation. For example, in Table 11.2, if arc-right were chosen at step 4, then the
 6055 parser would later be forced to attach the prepositional phrase with lox to the verb likes.
 6056 Note that the likes → bagels arc is indeed part of the correct dependency parse, but the
 6057 arc-standard transition system requires it to be created later in the derivation.

Beam search addresses this issue by maintaining a set of hypothetical derivations, called a beam. At step t of the derivation, there is a set of k hypotheses, each of which is a tuple of a score and a sequence of actions,

$$h_t^{(k)} = (s_t^{(k)}, A_t^{(k)}) \quad [11.23]$$

6058 Each hypothesis is then “expanded” by considering the set of all valid actions from the
 6059 current configuration $c_t^{(k)}$, written $\mathcal{A}(c_t^{(k)})$. This yields a large set of new hypotheses. For
 6060 each action $a\mathcal{A}(c_t^{(k)})$, we score the new hypothesis $A_t^{(k)} \oplus a$. The top k hypotheses by
 6061 this scoring metric are kept, and parsing proceeds to the next step (Zhang and Clark,
 6062 2008). Note that beam search requires a scoring function for action sequences, rather than
 6063 individual actions. This issue will be revisited in the next section.

6064 An example of beam search is shown in Figure 11.7, with a beam size of $K = 2$. For
 6065 the first transition, the only valid action is Shift, so there is only one possible configuration
 6066 at $t = 2$. From this configuration, there are three possible actions. The top two are
 6067 Arc-Right and Arc-Left, and so the resulting hypotheses from these actions are on the
 6068 beam at $t = 3$. From these configurations, there are three possible actions each, but the
 6069 best two are expansions of the bottom hypothesis at $t = 3$. Parsing continues until $t = 5$,
 6070 at which point both hypotheses reach an accepting state. The best-scoring hypothesis is
 6071 then selected as the parse.

6072 11.3.2 Scoring functions for transition-based parsers

Transition-based parsing requires selecting a series of actions. In greedy transition-based
 parsing, this can be done by training a classifier,

$$\hat{a} = \underset{a \in \mathcal{A}(c)}{\operatorname{argmax}} \Psi(a, c, \mathbf{w}; \boldsymbol{\theta}), \quad [11.24]$$

6073 where $\mathcal{A}(c)$ is the set of admissible actions in the current configuration c , \mathbf{w} is the input,
 6074 and Ψ is a scoring function with parameters $\boldsymbol{\theta}$ (Yamada and Matsumoto, 2003).

6075 A feature-based score can be computed, $\Psi(a, c, \mathbf{w}) = \boldsymbol{\theta} \cdot \mathbf{f}(a, c, \mathbf{w})$, using features that
 6076 may consider any aspect of the current configuration and input sequence. Typical features
 6077 for transition-based dependency parsing include: the word and part-of-speech of the top
 6078 element on the stack; the word and part-of-speech of the first, second, and third elements
 6079 on the input buffer; pairs and triples of words and parts-of-speech from the top of the stack
 6080 and the front of the buffer; the distance (in tokens) between the element on the top of the
 6081 stack and the element in the front of the input buffer; the number of modifiers of each of
 6082 these elements; and higher-order dependency features as described above in the section on
 6083 graph-based dependency parsing (see, e.g., Zhang and Nivre, 2011).

6084 Parse actions can also be scored by neural networks. For example, Chen and Manning
 6085 (2014) build a feedforward network in which the input layer consists of the concatenation
 6086 of embeddings of several words and tags:

- 6087 • the top three words on the stack, and the first three words on the buffer;
- 6088 • the first and second leftmost and rightmost children (dependents) of the top two
 words on the stack;
- 6089 • the leftmost and right most grandchildren of the top two words on the stack;
- 6090 • embeddings of the part-of-speech tags of these words.

Let us call this base layer $\mathbf{x}(c, \mathbf{w})$, defined as,

$$c = (\sigma, \beta, A)$$

$$\mathbf{x}(c, \mathbf{w}) = [\mathbf{v}_{w_{\sigma_1}}, \mathbf{v}_{t_{\sigma_1}}, \mathbf{v}_{w_{\sigma_2}}, \mathbf{v}_{t_{\sigma_2}}, \mathbf{v}_{w_{\sigma_3}}, \mathbf{v}_{t_{\sigma_3}}, \mathbf{v}_{w_{\beta_1}}, \mathbf{v}_{t_{\beta_1}}, \mathbf{v}_{w_{\beta_2}}, \mathbf{v}_{t_{\beta_2}}, \dots],$$

where $\mathbf{v}_{w_{\sigma_1}}$ is the embedding of the first word on the stack, $\mathbf{v}_{t_{\beta_2}}$ is the embedding of the part-of-speech tag of the second word on the buffer, and so on. Given this base encoding of the parser state, the score for the set of possible actions is computed through a feedforward network,

$$\mathbf{z} = g(\Theta^{(x \rightarrow z)} \mathbf{x}(c, \mathbf{w})) \quad [11.25]$$

$$\psi(a, c, \mathbf{w}; \boldsymbol{\theta}) = \Theta_a^{(z \rightarrow y)} \mathbf{z}, \quad [11.26]$$

6092 where the vector \mathbf{z} plays the same role as the features $f(a, c, \mathbf{w})$, but is a learned representation.
 6093 Chen and Manning (2014) use a cubic elementwise activation function, $g(x) = x^3$, so that
 6094 the hidden layer models products across all triples of input features. The learning algorithm
 6095 updates the embeddings as well as the parameters of the feedforward network.

6096 11.3.3 Learning to parse

6097 Transition-based dependency parsing suffers from a mismatch between the supervision,
 6098 which comes in the form of dependency trees, and the classifier’s prediction space, which
 6099 is a set of parsing actions. One solution is to create new training data by converting parse
 6100 trees into action sequences; another is to derive supervision directly from the parser’s
 6101 performance.

6102 11.3.3.1 Oracle-based training

6103 A transition system can be viewed as a function from action sequences (also called derivations)
 6104 to parse trees. The inverse of this function is a mapping from parse trees to derivations,
 6105 which is called an oracle. For the arc-standard and arc-eager parsing system, an oracle
 6106 can be computed in linear time in the length of the derivation (Kübler et al., 2009,
 6107 page 32). Both the arc-standard and arc-eager transition systems suffer from spurious
 6108 ambiguity: there exist dependency parses for which multiple derivations are possible, such
 6109 as $1 \leftarrow 2 \rightarrow 3$. The oracle must choose between these different derivations. For example,
 6110 the algorithm described by Kübler et al. (2009) would first create the left arc ($1 \leftarrow 2$),
 6111 and then create the right arc, $(1 \leftarrow 2) \rightarrow 3$; another oracle might begin by shifting twice,
 6112 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):

$$p(a | c, \mathbf{w}) = \frac{\exp \Psi(a, c, \mathbf{w}; \boldsymbol{\theta})}{\sum_{a' \in \mathcal{A}(c)} \exp \Psi(a', c, \mathbf{w}; \boldsymbol{\theta})} \quad [11.27]$$

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^N \sum_{t=1}^{|A^{(i)}|} \log p(a_t^{(i)} | c_t^{(i)}, \mathbf{w}), \quad [11.28]$$

where $|A^{(i)}|$ is the length of the action sequence $A^{(i)}$.

Recall that beam search requires a scoring function for action sequences. Such a score can be obtained by adding the log-likelihoods (or hinge losses) across all actions in the sequence (Chen and Manning, 2014).

11.3.3.2 Global objectives

The objective in Equation 11.27 is locally-normalized: it is the product of normalized probabilities over individual actions. A similar characterization could be made of non-probabilistic algorithms in which hinge-loss objectives are summed over individual actions. In either case, training on individual actions can be sub-optimal with respect to global performance, due to the label bias problem (Lafferty et al., 2001; Andor et al., 2016).

As a stylized example, suppose that a given configuration appears 100 times in the training data, with action a_1 as the oracle action in 51 cases, and a_2 as the oracle action in the other 49 cases. However, in cases where a_2 is correct, choosing a_1 results in a cascade of subsequent errors, while in cases where a_1 is correct, choosing a_2 results in only a single error. A classifier that is trained on a local objective function will learn to always choose a_1 , but choosing a_2 would minimize the overall number of errors.

This observation motivates a global objective, such as the globally-normalized conditional likelihood,

$$p(A^{(i)} | \mathbf{w}; \boldsymbol{\theta}) = \frac{\exp \sum_{t=1}^{|A^{(i)}|} \Psi(a_t^{(i)}, c_t^{(i)}, \mathbf{w})}{\sum_{A' \in \mathbb{A}(\mathbf{w})} \exp \sum_{t=1}^{|A'|} \Psi(a'_t, c'_t, \mathbf{w})}, \quad [11.29]$$

where the denominator sums over the set of all possible action sequences, $\mathbb{A}(\mathbf{w})$.⁵ In the conditional random field model for sequence labeling (§ 7.5.3), it was possible to compute this sum explicitly, using dynamic programming. In transition-based parsing, this is not possible. However, the sum can be approximated using beam search,

$$\sum_{A' \in \mathbb{A}(\mathbf{w})} \exp \sum_{t=1}^{|A'|} \Psi(a'_t, c'_t, \mathbf{w}) \approx \sum_{k=1}^K \exp \sum_{t=1}^{|A^{(k)}|} \Psi(a_t^{(k)}, c_t^{(k)}, \mathbf{w}), \quad [11.30]$$

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.

where $A^{(k)}$ is an action sequence on a beam of size K . This gives rise to the following loss function,

$$L(\boldsymbol{\theta}) = - \sum_{t=1}^{|A^{(i)}|} \Psi(a_t^{(i)}, c_t^{(i)}, \mathbf{w}) + \log \sum_{k=1}^K \exp \sum_{t=1}^{|A^{(k)}|} \Psi(a_t^{(k)}, c_t^{(k)}, \mathbf{w}). \quad [11.31]$$

6131 The derivatives of this loss involve expectations with respect to a probability distribution
 6132 over action sequences on the beam.

6133 11.3.3.3 *Early update and the incremental perceptron

6134 When learning in the context of beam search, the goal is to learn a decision function so that
 6135 the gold dependency parse is always reachable from at least one of the partial derivations
 6136 on the beam. (The combination of a transition system (such as beam search) and a scoring
 6137 function for actions is known as a policy.) To achieve this, we can make an early update
 6138 as soon as the oracle action sequence “falls off” the beam, even before a complete analysis
 6139 is available (Collins and Roark, 2004; Daumé III and Marcu, 2005). The loss can be based
 6140 on the best-scoring hypothesis on the beam, or the sum of all hypotheses (Huang et al.,
 6141 2012).

6142 For example, consider the beam search in Figure 11.7. In the correct parse, fish is
 6143 the head of dependency arcs to both of the other two words. In the arc-standard system,
 6144 this can be achieved only by using Shift for the first two actions. At $t = 3$, the oracle
 6145 action sequence has fallen off the beam. The parser should therefore stop, and update the
 6146 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
 6147 oracle sequence, and $\{A_{1:3}^{(k)}\}$ is the beam.

6148 This integration of incremental search and learning was first developed in the incremental
 6149 perceptron (Collins and Roark, 2004). This method updates the parameters with respect
 6150 to a hinge loss, which compares the top-scoring hypothesis and the gold action sequence,
 6151 up to the current point t . Several improvements to this basic protocol are possible:

- 6152 • As noted earlier, the gold dependency parse can be derived by multiple action
 6153 sequences. Rather than checking for the presence of a single oracle action sequence
 6154 on the beam, we can check if the gold dependency parse is reachable from the current
 6155 beam, using a dynamic oracle (Goldberg and Nivre, 2012).
- 6156 • By maximizing the score of the gold action sequence, we are training a decision
 6157 function to find the correct action given the gold context. But in reality, the parser
 6158 will make errors, and the parser is not trained to find the best action given a context
 6159 that may not itself be optimal. This issue is addressed by various generalizations of
 6160 incremental perceptron, known as learning to search (Daumé III et al., 2009). Some
 6161 of these methods are discussed in chapter 15.

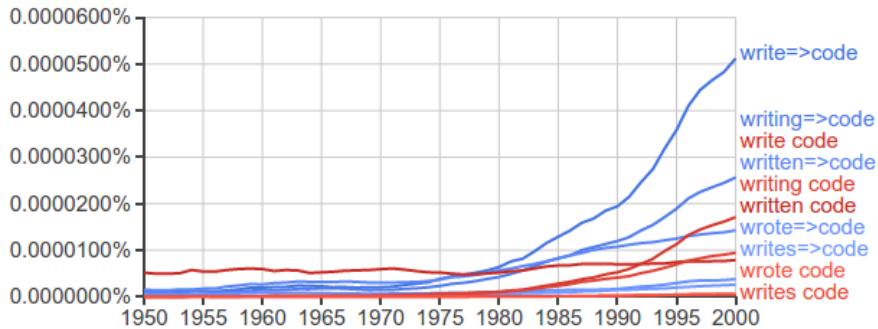


Figure 11.8: Google n-grams results for the bigram write code and the dependency arc write => code (and their morphological variants)

6162 11.4 Applications

6163 Dependency parsing is used in many real-world applications: any time you want to know
 6164 about pairs of words which might not be adjacent, you can use dependency arcs instead
 6165 of regular expression search patterns. For example, you may want to match strings like
 6166 delicious pastries, delicious French pastries, and the pastries are delicious.

6167 It is possible to search the Google *n*-grams corpus by dependency edges, finding the
 6168 trend in how often a dependency edge appears over time. For example, we might be
 6169 interested in knowing when people started talking about writing code, but we also want
 6170 write some code, write good code, write all the code, etc. The result of a search on the
 6171 dependency edge write → code is shown in Figure 11.8. This capability has been applied to
 6172 research in digital humanities, such as the analysis of gender in Shakespeare Muralidharan
 6173 and Hearst (2013).

A classic application of dependency parsing is relation extraction, which is described in chapter 17. The goal of relation extraction is to identify entity pairs, such as

(Melville, Moby-Dick)
 (Tolstoy, War and Peace)
 (Marqu  z, 100 Years of Solitude)
 (Shakespeare, A Midsummer Night's Dream),

6174 which stand in some relation to each other (in this case, the relation is authorship). Such
 6175 entity pairs are often referenced via consistent chains of dependency relations. Therefore,
 6176 dependency paths are often a useful feature in supervised systems which learn to detect new
 6177 instances of a relation, based on labeled examples of other instances of the same relation
 6178 type (Culotta and Sorensen, 2004; Fundel et al., 2007; Mintz et al., 2009).

6179 Cui et al. (2005) show how dependency parsing can improve automated question answering.
 6180 Suppose you receive the following query:

6181 (11.1) What percentage of the nation’s cheese does Wisconsin produce?

6182 The corpus contains this sentence:

6183 (11.2) In Wisconsin, where farmers produce 28% of the nation’s cheese, ...

6184 The location of Wisconsin in the surface form of this string makes it a poor match for the
 6185 query. However, in the dependency graph, there is an edge from produce to Wisconsin in
 6186 both the question and the potential answer, raising the likelihood that this span of text is
 6187 relevant to the question.

6188 A final example comes from sentiment analysis. As discussed in chapter 4, the polarity
 6189 of a sentence can be reversed by negation, e.g.

6190 (11.3) There is no reason at all to believe the polluters will suddenly become reasonable.

6191 By tracking the sentiment polarity through the dependency parse, we can better identify the
 6192 overall polarity of the sentence, determining when key sentiment words are reversed (Wilson
 6193 et al., 2005; Nakagawa et al., 2010).

6194 Additional resources

6195 More details on dependency grammar and parsing algorithms can be found in the manuscript
 6196 by Kübler et al. (2009). For a comprehensive but whimsical overview of graph-based
 6197 dependency parsing algorithms, see Eisner (1997). Jurafsky and Martin (2018) describe an
 6198 agenda-based version of beam search, in which the beam contains hypotheses of varying
 6199 lengths. New hypotheses are added to the beam only if their score is better than the
 6200 worst item currently on the beam. Another search algorithm for transition-based parsing
 6201 is easy-first, which abandons the left-to-right traversal order, and adds the highest-scoring
 6202 edges first, regardless of where they appear (Goldberg and Elhadad, 2010). Goldberg et al.
 6203 (2013) note that although transition-based methods can be implemented in linear time in
 6204 the length of the input, naïve implementations of beam search will require quadratic time,
 6205 due to the cost of copying each hypothesis when it is expanded on the beam. This issue
 6206 can be addressed by using a more efficient data structure for the stack.

6207 Exercises

- 6208 1. The dependency structure $1 \leftarrow 2 \rightarrow 3$, with 2 as the root, can be obtained from more
 6209 than one set of actions in arc-standard parsing. List both sets of actions that can
 6210 obtain this parse.

- 6211 2. This problem develops the relationship between dependency parsing and lexicalized
 6212 context-free parsing. Suppose you have a set of unlabeled arc scores $\{\psi(i \rightarrow j)\}_{i,j=1}^M \cup$
 6213 $\{\psi(\text{Root} \rightarrow j)\}_{j=1}^M$.
- 6214 a) Assuming each word type occurs no more than once in the input ($(i \neq j) \Rightarrow$
 6215 $(w_i \neq w_j)$), how would you construct a weighted lexicalized context-free grammar
 6216 so that the score of any projective dependency tree is equal to the score of some
 6217 equivalent derivation in the lexicalized context-free grammar?
- 6218 b) Verify that your method works for the example They fish.
- 6219 c) Does your method require the restriction that each word type occur no more
 6220 than once in the input? If so, why?
- 6221 d) *If your method required that each word type occur only once in the input,
 6222 show how to generalize it.
- 6223 3. In arc-factored dependency parsing of an input of length M , the score of a parse
 6224 is the sum of M scores, one for each arc. In second order dependency parsing, the
 6225 total score is the sum over many more terms. How many terms are the score of the
 6226 parse for Figure 11.2, using a second-order dependency parser with grandparent and
 6227 sibling features? Assume that a child of Root has no grandparent score, and that a
 6228 node with no siblings has no sibling scores.
- 6229 4. a) In the worst case, how many terms can be involved in the score of an input of
 6230 length M , assuming second-order dependency parsing? Describe the structure
 6231 of the worst-case parse. As in the previous problem, assume that there is only
 6232 one child of Root, and that it does not have any grandparent scores.
 6233 b) What about third-order dependency parsing?
- 6234 5. Provide the UD-style dependency parse for the sentence Xi-Lan eats shoots and
 6235 leaves, assuming leaves is a verb. Provide arc-standard and arc-eager derivations for
 6236 this dependency parse.
- 6237 6. Compute an upper bound on the number of successful derivations in arc-standard
 6238 shift-reduce parsing for unlabeled dependencies, as a function of the length of the
 6239 input, M . Hint: a lower bound is the number of projective decision trees, $\frac{1}{M+1} \binom{3M-2}{M-1}$ (?),
 6240 where $\binom{a}{b} = \frac{a!}{(a-b)!b!}$.
- 6241 7. In ??, we encounter the label bias problem, in which a decision may be locally correct,
 6242 yet lead to a cascade of errors. Design a scenario in which this occurs. Specifically:
 6243 • Assume an arc-standard dependency parser, whose action classifier considers
 6244 only the words at the top of the stack and at the front of the input buffer.
 6245 • Design two examples, which both involve a decision with identical features:

- 6246 – In one example, shift is the correct decision; in the other example, arc-left
6247 or arc-right is the correct decision.
6248 – In one of the two examples, a mistake should lead to a cascade of attachment
6249 errors.
6250 – In the other example, a mistake should lead only to a single attachment
6251 error.

6252

Part III

6253

Meaning

6254 Chapter 12

6255 Logical semantics

6256 The previous few chapters have focused on building systems that reconstruct the syntax of
6257 natural language — its structural organization — through tagging and parsing. But some
6258 of the most exciting and promising potential applications of language technology involve
6259 going beyond syntax to semantics — the underlying meaning of the text:

- 6260 • Answering questions, such as where is the nearest coffeeshop? or what is the middle
6261 name of the mother of the 44th President of the United States?.
- 6262 • Building a robot that can follow natural language instructions to execute tasks.
- 6263 • Translating a sentence from one language into another, while preserving the underlying
6264 meaning.
- 6265 • Fact-checking an article by searching the web for contradictory evidence.
- 6266 • Logic-checking an argument by identifying contradictions, ambiguity, and unsupported
6267 assertions.

6268 Semantic analysis involves converting natural language into a meaning representation.
6269 To be useful, a meaning representation must meet several criteria:

- 6270 • c1: it should be unambiguous: unlike natural language, there should be exactly one
6271 meaning per statement;
- 6272 • c2: it should provide a way to link language to external knowledge, observations, and
6273 actions;
- 6274 • c3: it should support computational inference, so that meanings can be combined to
6275 derive additional knowledge;
- 6276 • c4: it should be expressive enough to cover the full range of things that people talk
6277 about in natural language.

6278 Much more than this can be said about the question of how best to represent knowledge
 6279 for computation (e.g., Sowa, 2000), but this chapter will focus on these four criteria.

6280 12.1 Meaning and denotation

6281 The first criterion for a meaning representation is that statements in the representation
 6282 should be unambiguous — they should have only one possible interpretation. Natural
 6283 language does not have this property: as we saw in chapter 10, sentences like cats scratch
 6284 people with claws have multiple interpretations.

6285 But what does it mean for a statement to be unambiguous? Programming languages
 6286 provide a useful example: the output of a program is completely specified by the rules
 6287 of the language and the properties of the environment in which the program is run. For
 6288 example, the python code $5 + 3$ will have the output 8, as will the codes $(4*4)-(3*3)+1$
 6289 and $((8))$. This output is known as the denotation of the program, and can be written as,

$$\llbracket 5+3 \rrbracket = \llbracket (4*4)-(3*3)+1 \rrbracket = \llbracket ((8)) \rrbracket = 8. \quad [12.1]$$

6290 The denotations of these arithmetic expressions are determined by the meaning of the
 6291 constants (e.g., 5, 3) and the relations (e.g., +, *, (,)). Now let's consider another snippet
 6292 of python code, `double(4)`. The denotation of this code could be, $\llbracket \text{double}(4) \rrbracket = 8$, or it
 6293 could be $\llbracket \text{double}(4) \rrbracket = 44$ — it depends on the meaning of `double`. This meaning is defined
 6294 in a world model \mathcal{M} as an infinite set of pairs. We write the denotation with respect to
 6295 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 model would also
 6296 define the (infinite) list of constants, e.g., $\{0,1,2,\dots\}$. As long as the denotation of string ϕ
 6297 in model \mathcal{M} can be computed unambiguously, the language can be said to be unambiguous.

6298 This approach to meaning is known as model-theoretic semantics, and it addresses not
 6299 only criterion *c1* (no ambiguity), but also *c2* (connecting language to external knowledge,
 6300 observations, and actions). For example, we can connect a representation of the meaning
 6301 of a statement like the capital of Georgia with a world model that includes knowledge
 6302 base of geographical facts, obtaining the denotation Atlanta. We might populate a world
 6303 model by applying an image analysis algorithm to Figure 12.1, and then use this world
 6304 model to evaluate propositions like a man is riding a moose. Another desirable property of
 6305 model-theoretic semantics is that when the facts change, the denotations change too: the
 6306 meaning representation of President of the USA would have a different denotation in the
 6307 model \mathcal{M}_{2014} as it would in \mathcal{M}_{2022} .

6308 12.2 Logical representations of meaning

6309 Criterion *c3* requires that the meaning representation support inference — for example,
 6310 automatically deducing new facts from known premises. While many representations have



Figure 12.1: A (doctored) image, which could be the basis for a world model

been proposed that meet these criteria, the most mature is the language of first-order logic.¹

12.2.1 Propositional logic

The bare bones of logical meaning representation are Boolean operations on propositions:

Propositional symbols. Greek symbols like ϕ and ψ will be used to represent propositions, which are statements that are either true or false. For example, ϕ may correspond to the proposition, bagels are delicious.

Boolean operators. We can build up more complex propositional formulas from Boolean operators. These include:

- Negation $\neg\phi$, which is true if ϕ is false.
- Conjunction, $\phi \wedge \psi$, which is true if both ϕ and ψ are true.
- Disjunction, $\phi \vee \psi$, which is true if at least one of ϕ and ψ is true
- Implication, $\phi \Rightarrow \psi$, which is true unless ϕ is true and ψ is false. Implication has identical truth conditions to $\neg\phi \vee \psi$.

¹Alternatives include the “variable-free” representation used in semantic parsing of geographical queries (Zelle and Mooney, 1996) and robotic control (Ge and Mooney, 2005), and dependency-based compositional semantics (Liang et al., 2013).

- 6325 • Equivalence, $\phi \Leftrightarrow \psi$, which is true if ϕ and ψ are both true or both false.
 6326 Equivalence has identical truth conditions to $(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$.

6327 It is not strictly necessary to have all five Boolean operators: readers familiar with
 6328 Boolean logic will know that it is possible to construct all other operators from either the
 6329 NAND (not-and) or NOR (not-or) operators. Nonetheless, it is clearest to use all five
 6330 operators. From the truth conditions for these operators, it is possible to define a number
 6331 of “laws” for these Boolean operators, such as,

- 6332 • Commutativity: $\phi \wedge \psi = \psi \wedge \phi$, $\phi \vee \psi = \psi \vee \phi$
 6333 • Associativity: $\phi \wedge (\psi \wedge \chi) = (\phi \wedge \psi) \wedge \chi$, $\phi \vee (\psi \vee \chi) = (\phi \vee \psi) \vee \chi$
 6334 • Complementation: $\phi \wedge \neg\phi = \perp$, $\phi \vee \neg\phi = \top$, where \top indicates a true proposition
 6335 and \perp indicates a false proposition.

These laws can be combined to derive further equivalences, which can support logical inferences. For example, suppose $\phi =$ The music is loud and $\psi =$ Max can't sleep. Then if we are given,

$$\begin{aligned} \phi \Rightarrow \psi & \quad \text{If the music is loud, Max can't sleep.} \\ \phi & \quad \text{The music is loud.} \end{aligned}$$

6336 we can derive ψ (Max can't sleep) by application of modus ponens, which is one of a
 6337 set of inference rules that can be derived from more basic laws and used to manipulate
 6338 propositional formulas. Automated theorem provers are capable of applying inference rules
 6339 to a set of premises to derive desired propositions (Loveland, 2016).

6340 12.2.2 First-order logic

6341 Propositional logic is so named because it treats propositions as its base units. However,
 6342 the criterion *c4* states that our meaning representation should be sufficiently expressive.
 6343 Now consider the sentence pair,

- 6344 (12.1) If anyone is making noise, then Max can't sleep.
 6345 Abigail is making noise.

6346 People are capable of making inferences from this sentence pair, but such inferences require
 6347 formal tools that are beyond propositional logic. To understand the relationship between
 6348 the statement anyone is making noise and the statement Abigail is making noise, our
 6349 meaning representation requires the additional machinery of first-order logic (FOL).

6350 In FOL, logical propositions can be constructed from relationships between entities.
 6351 Specifically, FOL extends propositional logic with the following classes of terms:

6352 Constants. These are elements that name individual entities in the model, such as max
 6353 and abigail. The denotation of each constant in a model \mathcal{M} is an element in the
 6354 model, e.g., $\llbracket \text{max} \rrbracket = m$ and $\llbracket \text{abigail} \rrbracket = a$.

6355 Relations. Relations can be thought of as sets of entities, or sets of tuples. For example, the
 6356 relation can-sleep is defined as the set of entities who can sleep, and has the denotation
 6357 $\llbracket \text{can-sleep} \rrbracket = \{a, m, \dots\}$. To test the truth value of the proposition $\text{can-sleep}(\text{max})$,
 6358 we ask whether $\llbracket \text{max} \rrbracket \in \llbracket \text{can-sleep} \rrbracket$. Logical relations that are defined over sets of
 6359 entities are sometimes called properties.

6360 Relations may also be ordered tuples of entities. For example $\text{brother}(\text{max}, \text{abigail})$
 6361 expresses the proposition that max is the brother of abigail. The denotation of such
 6362 relations is a set of tuples, $\llbracket \text{brother} \rrbracket = \{(m, a), (x, y), \dots\}$. To test the truth value of
 6363 the proposition $\text{brother}(\text{max}, \text{abigail})$, we ask whether the tuple $(\llbracket \text{max} \rrbracket, \llbracket \text{abigail} \rrbracket)$ is
 6364 in the denotation $\llbracket \text{brother} \rrbracket$.

Using constants and relations, it is possible to express statements like Max can't sleep and Max is Abigail's brother:

$$\neg \text{can-sleep}(\text{max}) \\ \text{brother}(\text{max}, \text{abigail}).$$

These statements can also be combined using Boolean operators, such as,

$$(\text{brother}(\text{max}, \text{abigail}) \vee \text{brother}(\text{max}, \text{steve})) \Rightarrow \neg \text{can-sleep}(\text{max}).$$

6365 This fragment of first-order logic permits only statements about specific entities. To
 6366 support inferences about statements like If anyone is making noise, then Max can't sleep,
 6367 two more elements must be added to the meaning representation:

6368 Variables. Variables are mechanisms for referring to entities that are not locally specified.
 6369 We can then write $\text{can-sleep}(x)$ or $\text{brother}(x, \text{abigail})$. In these cases, x is a free
 6370 variable, meaning that we have not committed to any particular assignment.

6371 Quantifiers. Variables are bound by quantifiers. There are two quantifiers in first-order
 6372 logic.²

²In first-order logic, it is possible to quantify only over entities. In second-order logic, it is possible to quantify over properties, supporting statements like Butch has every property that a good boxer has (example from Blackburn and Bos, 2005),

$$\forall P \forall x ((\text{good-boxer}(x) \Rightarrow P(x)) \Rightarrow P(\text{butch})).$$

[12.2]

- 6373 • The existential quantifier \exists , which indicates that there must be at least one
 6374 entity to which the variable can bind. For example, the statement $\exists x \text{makes-noise}(x)$
 6375 indicates that there is at least one entity for which makes-noise is true.
 6376 • The universal quantifier \forall , which indicates that the variable must be able to
 6377 bind to any entity in the model. For example, the statement,

$$\text{makes-noise}(\text{abigail}) \Rightarrow (\forall x \neg \text{can-sleep}(x)) \quad [12.3]$$

6378 asserts that if Abigail makes noise, no one can sleep.

6379 The expressions $\exists x$ and $\forall x$ make x into a bound variable. A formula that contains
 6380 no free variables is a sentence.

6381 Functions. Functions map from entities to entities, e.g., $\llbracket \text{capital-of}(\text{georgia}) \rrbracket = \llbracket \text{atlanta} \rrbracket$.
 6382 With functions, it is convenient to add an equality operator, supporting statements
 6383 like,

$$\forall x \exists y \text{mother-of}(x) = \text{daughter-of}(y). \quad [12.4]$$

6384 Note that mother-of is a functional analogue of the relation mother, so that $\text{mother-of}(x) = y$
 6385 if $\text{mother}(x, y)$. Any logical formula that uses functions can be rewritten using only
 6386 relations and quantification. For example,

$$\text{makes-noise}(\text{mother-of}(\text{abigail})) \quad [12.5]$$

6387 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:

$$\forall x \exists y \text{ speaks}(x, y) \quad [12.6]$$

$$\exists y \forall x \text{ speaks}(x, y). \quad [12.7]$$

6388 In the first case, y may refer to several different languages, while in the second case, there
 6389 is a single y that is spoken by everyone.

6390 12.2.2.1 Truth-conditional semantics

6391 One way to look at the meaning of an FOL sentence ϕ is as a set of truth conditions,
 6392 or models under which ϕ is satisfied. But how to determine whether a sentence is true
 6393 or false in a given model? We will approach this inductively, starting with a predicate
 6394 applied to a tuple of constants. The truth of such a sentence depends on whether the

6395 tuple of denotations of the constants is in the denotation of the predicate. For example,
 6396 $\text{capital}(\text{georgia}, \text{atlanta})$ is true in model \mathcal{M} iff,

$$([\![\text{georgia}]\!]_{\mathcal{M}}, [\![\text{atlanta}]\!]_{\mathcal{M}}) \in [\![\text{capital}]\!]_{\mathcal{M}}. \quad [12.8]$$

6397 The Boolean operators \wedge, \vee, \dots provide ways to construct more complicated sentences,
 6398 and the truth of such statements can be assessed based on the truth tables associated with
 6399 these operators. The statement $\exists x\phi$ is true if there is some assignment of the variable x
 6400 to an entity in the model such that ϕ is true; the statement $\forall x\phi$ is true if ϕ is true under
 6401 all possible assignments of x . More formally, we would say that ϕ is satisfied under \mathcal{M} ,
 6402 written as $\mathcal{M} \models \phi$.

6403 Truth conditional semantics allows us to define several other properties of sentences and
 6404 pairs of sentences. Suppose that in every \mathcal{M} under which ϕ is satisfied, another formula ψ
 6405 is also satisfied; then ϕ entails ψ , which is also written as $\phi \models \psi$. For example,

$$\text{capital}(\text{georgia}, \text{atlanta}) \models \exists x \text{capital}(\text{georgia}, x). \quad [12.9]$$

6406 A statement that is satisfied under any model, such as $\phi \vee \neg\phi$, is valid, written $\models (\phi \vee \neg\phi)$.
 6407 A statement that is not satisfied under any model, such as $\phi \wedge \neg\phi$, is unsatisfiable, or
 6408 inconsistent. A model checker is a program that determines whether a sentence ϕ is satisfied
 6409 in \mathcal{M} . A model builder is a program that constructs a model in which ϕ is satisfied.
 6410 The problems of checking for consistency and validity in first-order logic are undecidable,
 6411 meaning that there is no algorithm that can automatically determine whether an FOL
 6412 formula is valid or inconsistent.

6413 12.2.2.2 Inference in first-order logic

6414 Our original goal was to support inferences that combine general statements If anyone is
 6415 making noise, then Max can't sleep with specific statements like Abigail is making noise.
 6416 We can now represent such statements in first-order logic, but how are we to perform
 6417 the inference that Max can't sleep? One approach is to use "generalized" versions of
 6418 propositional inference rules like modus ponens, which can be applied to FOL formulas.
 6419 By repeatedly applying such inference rules to a knowledge base of facts, it is possible
 6420 to produce proofs of desired propositions. To find the right sequence of inferences to
 6421 derive a desired theorem, classical artificial intelligence search algorithms like backward
 6422 chaining can be applied. Such algorithms are implemented in interpreters for the prolog
 6423 logic programming language (Pereira and Shieber, 2002).

6424 12.3 Semantic parsing and the lambda calculus

6425 The previous section laid out a lot of formal machinery; the remainder of this chapter links
 6426 these formalisms back to natural language. Given an English sentence like Alex likes Brit,

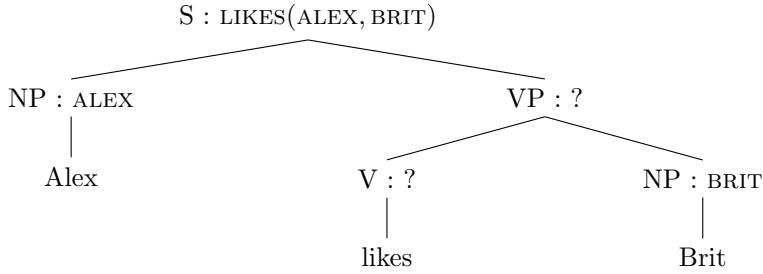


Figure 12.2: The principle of compositionality requires that we identify meanings for the constituents *likes* and *likes Brit* that will make it possible to compute the meaning for the entire sentence.

6427 how can we obtain the desired first-order logical representation, *likes(alex,brit)*? This is the
 6428 task of semantic parsing. Just as a syntactic parser is a function from a natural language
 6429 sentence to a syntactic structure such as a phrase structure tree, a semantic parser is a
 6430 function from natural language to logical formulas.

6431 As in syntactic analysis, semantic parsing is difficult because the space of inputs and
 6432 outputs is very large, and their interaction is complex. Our best hope is that, like syntactic
 6433 parsing, semantic parsing can somehow be decomposed into simpler sub-problems. This
 6434 idea, usually attributed to the German philosopher Gottlob Frege, is called the principle
 6435 of compositionality: the meaning of a complex expression is a function of the meanings of
 6436 that expression's constituent parts. We will define these “constituent parts” as syntactic
 6437 constituents: noun phrases and verb phrases. These constituents are combined using
 6438 function application: if the syntactic parse contains the production $x \rightarrow y z$, then the
 6439 semantics of x , written $x.\text{sem}$, will be computed as a function of the semantics of the
 6440 constituents, $y.\text{sem}$ and $z.\text{sem}$.³ ⁴

6441 12.3.1 The lambda calculus

6442 Let's see how this works for a simple sentence like *Alex likes Brit*, whose syntactic structure
 6443 is shown in Figure 12.2. Our goal is the formula, *likes(alex,brit)*, and it is clear that the
 6444 meaning of the constituents *Alex* and *Brit* should be *alex* and *brit*. That leaves two more
 6445 constituents: the verb *likes*, and the verb phrase *likes Brit*. The meanings of these units

³§ 9.3.2 briefly discusses Combinatory Categorial Grammar (CCG) as an alternative to a phrase-structure analysis of syntax. CCG is argued to be particularly well-suited to semantic parsing (Hockenmaier and Steedman, 2007), and is used in much of the contemporary work on machine learning for semantic parsing, summarized in § 12.4.

⁴The approach of algorithmically building up meaning representations from a series of operations on the syntactic structure of a sentence is generally attributed to the philosopher Richard Montague, who published a series of influential papers on the topic in the early 1970s (e.g., Montague, 1973).

must be defined in a way that makes it possible to recover the desired meaning for the entire sentence by function application. If the meanings of Alex and Brit are constants, then the meanings of likes and likes Brit must be functional expressions, which can be applied to their siblings to produce the desired analyses.

Modeling these partial analyses requires extending the first-order logic meaning representation. We do this by adding lambda expressions, which are descriptions of anonymous functions,⁵ e.g.,

$$\lambda x.\text{likes}(x, \text{brit}). \quad [12.10]$$

This functional expression is the meaning of the verb phrase likes Brit; it takes a single argument, and returns the result of substituting that argument for x in the expression $\text{likes}(x, \text{brit})$. We write this substitution as,

$$(\lambda x.\text{likes}(x, \text{brit}))@\text{alex} = \text{likes}(\text{alex}, \text{brit}), \quad [12.11]$$

with the symbol “@” indicating function application. Function application in the lambda calculus is sometimes called β -reduction or β -conversion. The expression $\phi@\psi$ indicates a function application to be performed by β -reduction, and $\phi(\psi)$ indicates a function or predicate in the final logical form.

Equation 12.11 shows how to obtain the desired semantics for the sentence Alex likes Brit: by applying the lambda expression $\lambda x.\text{likes}(x, \text{brit})$ to the logical constant alex. This rule of composition can be specified in a syntactic-semantic grammar, in which syntactic productions are paired with semantic operations. For the syntactic production $S \rightarrow NP VP$, we have the semantic rule $VP.sem@NP.sem$.

The meaning of the transitive verb phrase likes Brit can also be obtained by function application on its syntactic constituents. For the syntactic production $VP \rightarrow V NP$, we apply the semantic rule,

$$VP.sem = (V.sem)@NP.sem \quad [12.12]$$

$$= (\lambda y. \lambda x. \text{likes}(x, y)) @ (\text{brit}) \quad [12.13]$$

$$= \lambda x. \text{likes}(x, \text{brit}). \quad [12.14]$$

Thus, the meaning of the transitive verb likes is a lambda expression whose output is another lambda expression: it takes y as an argument to fill in one of the slots in the likes relation, and returns a lambda expression that is ready to take an argument to fill in the other slot.⁶

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.

⁶This can be written in a few different ways. The notation $\lambda y, x. \text{likes}(x, y)$ is a somewhat informal way to indicate a lambda expression that takes two arguments; this would be acceptable in functional programming. Logicians (e.g., Carpenter, 1997) often prefer the more formal notation $\lambda y. \lambda x. \text{likes}(x)(y)$, indicating that each lambda expression takes exactly one argument.

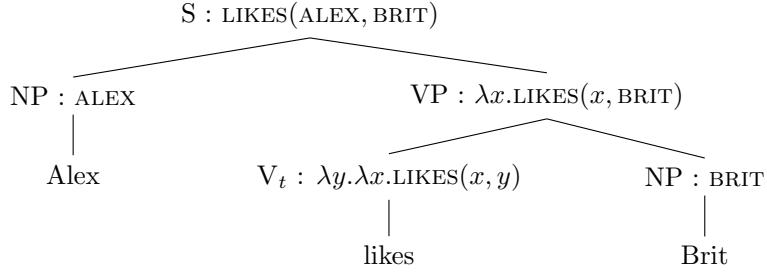


Figure 12.3: Derivation of the semantic representation for Alex likes Brit in the grammar G_1 .

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

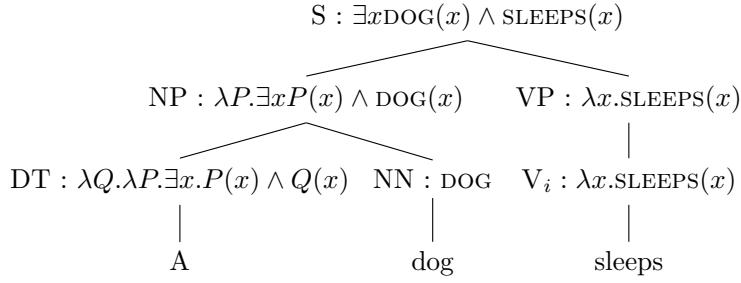
Table 12.1: G_1 , a minimal syntactic-semantic context-free grammar

6469 Table 12.1 shows a minimal syntactic-semantic grammar fragment, G_1 . The complete
 6470 derivation of Alex likes Brit in G_1 is shown in Figure 12.3. In addition to the transitive
 6471 verb likes, the grammar also includes the intransitive verb sleeps; it should be clear how
 6472 to derive the meaning of sentences like Alex sleeps. For verbs that can be either transitive
 6473 or intransitive, such as eats, we would have two terminal productions, one for each sense
 6474 (terminal productions are also called the lexical entries). Indeed, most of the grammar is
 6475 in the lexicon (the terminal productions), since these productions select the basic units of
 6476 the semantic interpretation.

6477 12.3.2 Quantification

6478 Things get more complicated when we move from sentences about named entities to
 6479 sentences that involve more general noun phrases. Let's consider the example, A dog
 6480 sleeps, which has the meaning $\exists x \text{dog}(x) \wedge \text{sleeps}(x)$. Clearly, the dog relation will be
 6481 introduced by the word dog, and the sleep relation will be introduced by the word sleeps.
 6482 The existential quantifier \exists must be introduced by the lexical entry for the determiner a.⁷

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

Figure 12.4: Derivation of the semantic representation for A dog sleeps, in grammar G_2

6483 However, this seems problematic for the compositional approach taken in the grammar G_1 :
 6484 if the semantics of the noun phrase a dog is an existentially quantified expression, how
 6485 can it be the argument to the semantics of the verb sleeps, which expects an entity? And
 6486 where does the logical conjunction come from?

6487 There are a few different approaches to handling these issues.⁸ We will begin by
 6488 reversing the semantic relationship between subject NPs and VPs, so that the production
 6489 $S \rightarrow NP\ VP$ has the semantics $NP.sem @ VP.sem$: the meaning of the sentence is now the
 6490 semantics of the noun phrase applied to the verb phrase. The implications of this change
 6491 are best illustrated by exploring the derivation of the example, shown in Figure 12.4. Let's
 6492 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 adjacent noun, which in this case is dog. Thus, the noun phrase a dog has the semantics,

$$NP.sem = Det.sem @ Nn.sem \quad [12.16]$$

$$= (\lambda P. \lambda Q. \exists x P(x) \wedge Q(x)) @ (dog) \quad [12.17]$$

$$= \lambda Q. \exists x \text{dog}(x) \wedge Q(x). \quad [12.18]$$

6493 This is a lambda expression that is expecting another relation, Q , which will be provided
 6494 by the verb phrase, sleeps. This gives the desired analysis, $\exists x \text{dog}(x) \wedge \text{sleeps}(x)$.⁹

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.

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.

6495 If noun phrases like a dog are interpreted as lambda expressions, then proper nouns like
 6496 Alex must be treated in the same way. This is achieved by type-raising from constants to
 6497 lambda expressions, $x \Rightarrow \lambda P.P(x)$. After type-raising, the semantics of Alex is $\lambda P.P(\text{alex})$
 6498 — a lambda expression that expects a relation to tell us something about alex.¹⁰ Again,
 6499 make sure you see how the analysis in Figure 12.4 can be applied to the sentence Alex
 6500 sleeps.

6501 Direct objects are handled by applying the same type-raising operation to transitive
 6502 verbs: the meaning of verbs such as likes is raised to,

$$\lambda P.\lambda x.P(\lambda y.\text{likes}(x, y)) \quad [12.19]$$

As a result, we can keep the verb phrase production $\text{VP.sem} = \text{V.sem}@\text{NP.sem}$, knowing that the direct object will provide the function P in Equation 12.19. To see how this works, let's analyze the verb phrase likes a dog. After uniquely relabeling each lambda variable, we have,

$$\begin{aligned} \text{VP.sem} &= \text{V.sem}@\text{NP.sem} \\ &= (\lambda P.\lambda x.P(\lambda y.\text{likes}(x, y))) @ (\lambda Q.\exists z \text{dog}(z) \wedge Q(z)) \\ &= \lambda x.(\lambda Q.\exists z \text{dog}(z) \wedge Q(z)) @ (\lambda y.\text{likes}(x, y)) \\ &= \lambda x.\exists z \text{dog}(z) \wedge (\lambda y.\text{likes}(x, y)) @ z \\ &= \lambda x.\exists z \text{dog}(z) \wedge \text{likes}(x, z). \end{aligned}$$

6503 These changes are summarized in the revised grammar G_2 , shown in Table 12.2.
 6504 Figure 12.5 shows a derivation that involves a transitive verb, an indefinite noun phrase,
 6505 and a proper noun.

6506 12.4 Learning semantic parsers

6507 As with syntactic parsing, any syntactic-semantic grammar with sufficient coverage risks
 6508 producing many possible analyses for any given sentence. Machine learning is the dominant
 6509 approach to selecting a single analysis. We will focus on algorithms that learn to score
 6510 logical forms by attaching weights to features of their derivations (Zettlemoyer and Collins,
 6511 2005). Alternative approaches include transition-based parsing (Zelle and Mooney, 1996;
 6512 Misra and Artzi, 2016) and methods inspired by machine translation (Wong and Mooney,

¹⁰Compositional semantic analysis is often supported by type systems, which make it possible to check whether a given function application is valid. The base types are entities e and truth values t . A property, such as dog, is a function from entities to truth values, so its type is written $\langle e, t \rangle$. A transitive verb has type $\langle e, \langle e, t \rangle \rangle$: after receiving the first entity (the direct object), it returns a function from entities to truth values, which will be applied to the subject of the sentence. The type-raising operation $x \Rightarrow \lambda P.P(x)$ corresponds to a change in type from e to $\langle \langle e, t \rangle, t \rangle$: it expects a function from entities to truth values, and returns a truth value.

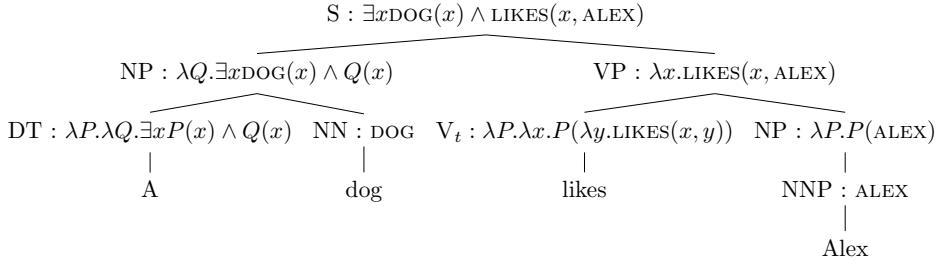


Figure 12.5: Derivation of the semantic representation for A dog likes Alex.

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	$\lambda P. P(Nnp.sem)$
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	$\lambda P. \lambda x. P(\lambda y. likes(x, y))$
V _i	\rightarrow sleeps	$\lambda x. sleeps(x)$
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

6513 2006). Methods also differ in the form of supervision used for learning, which can range
 6514 from complete derivations to much more limited training signals. We will begin with the
 6515 case of complete supervision, and then consider how learning is still possible even when
 6516 seemingly key information is missing.

6517 Datasets Early work on semantic parsing focused on natural language expressions of
 6518 geographical database queries, such as What states border Texas. The GeoQuery dataset
 6519 of Zelle and Mooney (1996) was originally coded in prolog, but has subsequently been
 6520 expanded and converted into the SQL database query language by Popescu et al. (2003)
 6521 and into first-order logic with lambda calculus by Zettlemoyer and Collins (2005), providing
 6522 logical forms like $\lambda x. state(x) \wedge borders(x, \text{texas})$. Another early dataset consists of instructions
 6523 for RoboCup robot soccer teams (Kate et al., 2005). More recent work has focused
 6524 on broader domains, such as the Freebase database (Bollacker et al., 2008), for which

queries have been annotated by Krishnamurthy and Mitchell (2012) and Cai and Yates (2013). Other recent datasets include child-directed speech (Kwiatkowski et al., 2012) and elementary school science exams (Krishnamurthy, 2016).

12.4.1 Learning from derivations

Let $\mathbf{w}^{(i)}$ indicate a sequence of text, and let $\mathbf{y}^{(i)}$ indicate the desired logical form. For example:

$$\begin{aligned}\mathbf{w}^{(i)} &= \text{Alex eats shoots and leaves} \\ \mathbf{y}^{(i)} &= \text{eats(alex,shoots) } \wedge \text{eats(alex,leaves)}\end{aligned}$$

In the standard supervised learning paradigm that was introduced in § 2.2, we first define a feature function, $\mathbf{f}(\mathbf{w}, \mathbf{y})$, and then learn weights on these features, so that $\mathbf{y}^{(i)} = \operatorname{argmax}_{\mathbf{y}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \mathbf{y})$. The weight vector $\boldsymbol{\theta}$ is learned by comparing the features of the true label $\mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})$ against either the features of the predicted label $\mathbf{f}(\mathbf{w}^{(i)}, \hat{\mathbf{y}})$ (perceptron, support vector machine) or the expected feature vector $E_{\mathbf{y}|\mathbf{w}}[\mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y})]$ (logistic regression).

While this basic framework seems similar to discriminative syntactic parsing, there is a crucial difference. In (context-free) syntactic parsing, the annotation $\mathbf{y}^{(i)}$ 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 eats(alex,shoots) \wedge eats(alex,leaves) does not reveal the syntactic-semantic productions that were used to obtain it. Indeed, there may be spurious ambiguity, so that a single logical form can be reached by multiple derivations. (We previously encountered spurious ambiguity in transition-based dependency parsing, § 11.3.2.)

These ideas can be formalized by introducing an additional variable \mathbf{z} , representing the derivation of the logical form \mathbf{y} from the text \mathbf{w} . Assume that the feature function decomposes across the productions in the derivation, $\mathbf{f}(\mathbf{w}, \mathbf{z}, \mathbf{y}) = \sum_{t=1}^T \mathbf{f}(\mathbf{w}, z_t, \mathbf{y})$, where z_t indicates a single syntactic-semantic production. For example, we might have a feature for the production $S \rightarrow NP VP : NP.sem @ VP.sem$, as well as for terminal productions like $Nnp \rightarrow Alex : alex$. Under this decomposition, it is possible to compute scores for each semantically-annotated subtree in the analysis of \mathbf{w} , so that bottom-up parsing algorithms like CKY (§ 10.1) can be applied to find the best-scoring semantic analysis.

Figure 12.6 shows a derivation of the correct semantic analysis of the sentence Alex eats shoots and leaves, in a simplified grammar in which the plural noun phrases shoots and leaves are interpreted as logical constants shoots and leaves_n. Figure 12.7 shows a derivation of an incorrect analysis. Assuming one feature per production, the perceptron update is shown in Table 12.3. From this update, the parser would learn to prefer the noun interpretation of leaves over the verb interpretation. It would also learn to prefer noun phrase coordination over verb phrase coordination.

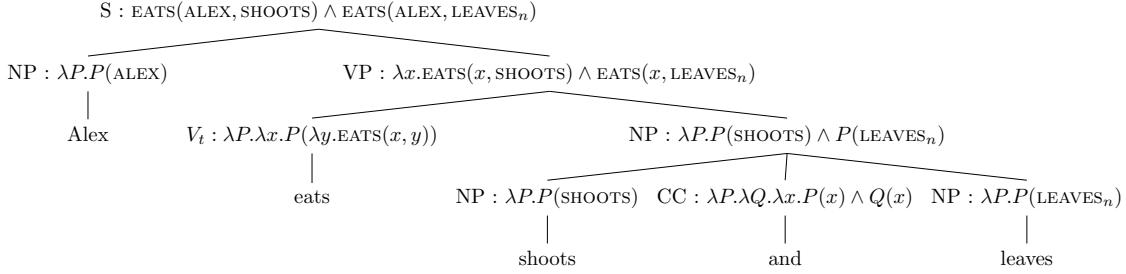


Figure 12.6: Derivation for gold semantic analysis of Alex eats shoots and leaves

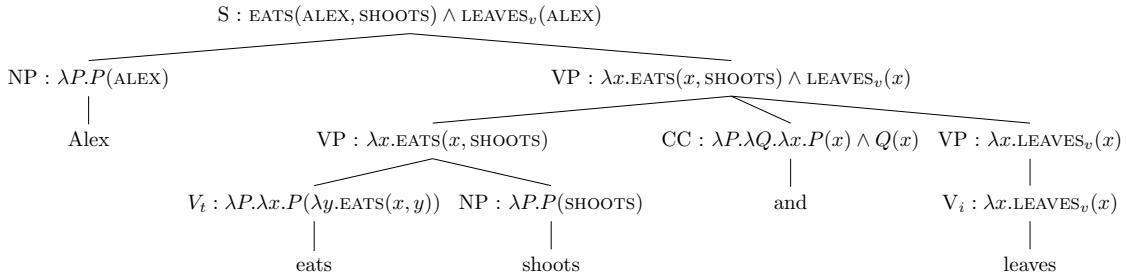


Figure 12.7: Derivation for incorrect semantic analysis of Alex eats shoots and leaves

$NP_1 \rightarrow NP_2 \text{ Cc } NP_3$	$(\text{Cc.sem} @ (\text{NP}_2.\text{sem})) @ (\text{NP}_3.\text{sem})$	+1
$VP_1 \rightarrow VP_2 \text{ Cc } VP_3$	$(\text{Cc.sem} @ (\text{VP}_2.\text{sem})) @ (\text{VP}_3.\text{sem})$	-1
$NP \rightarrow \text{leaves}$	leaves_n	+1
$VP \rightarrow V_i$	$V_i.\text{sem}$	-1
$V_i \rightarrow \text{leaves}$	$\lambda x.\text{leaves}_v$	-1

Table 12.3: Perceptron update for analysis in Figure 12.6 (gold) and Figure 12.7 (predicted)

While the update is explained in terms of the perceptron, it would be easy to replace the perceptron with a conditional random field. In this case, the online updates would be based on feature expectations, which can be computed using the inside-outside algorithm (§ 10.6).

12.4.2 Learning from logical forms

Complete derivations are expensive to annotate, and are rarely available.¹¹ One solution is to focus on learning from logical forms directly, while treating the derivations as latent

¹¹An exception is the work of Ge and Mooney (2005), who annotate the meaning of each syntactic constituents for several hundred sentences.

variables (Zettlemoyer and Collins, 2005). In a conditional probabilistic model over logical forms \mathbf{y} and derivations \mathbf{z} , we have,

$$p(\mathbf{y}, \mathbf{z} | \mathbf{w}) = \frac{\exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \mathbf{z}, \mathbf{y}))}{\sum_{\mathbf{y}', \mathbf{z}'} \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \mathbf{z}', \mathbf{y}'))}, \quad [12.20]$$

which is the standard log-linear model, applied to the logical form \mathbf{y} and the derivation \mathbf{z} .

Since the derivation \mathbf{z} unambiguously determines the logical form \mathbf{y} , it may seem silly to model the joint probability over \mathbf{y} and \mathbf{z} . However, since \mathbf{z} is unknown, it can be marginalized out,

$$p(\mathbf{y} | \mathbf{w}) = \sum_{\mathbf{z}} p(\mathbf{y}, \mathbf{z} | \mathbf{w}). \quad [12.21]$$

The semantic parser can then select the logical form with the maximum log marginal probability,

$$\log \sum_{\mathbf{z}} p(\mathbf{y}, \mathbf{z} | \mathbf{w}) = \log \sum_{\mathbf{z}} \frac{\exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \mathbf{z}, \mathbf{y}))}{\sum_{\mathbf{y}', \mathbf{z}'} \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \mathbf{z}', \mathbf{y}'))} \quad [12.22]$$

$$\propto \log \sum_{\mathbf{z}} \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \mathbf{z}', \mathbf{y}')) \quad [12.23]$$

$$\geq \max_{\mathbf{z}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \mathbf{z}, \mathbf{y}). \quad [12.24]$$

It is impossible to push the log term inside the sum over \mathbf{z} , so our usual linear scoring function does not apply. We can recover this scoring function only in approximation, by taking the max (rather than the sum) over derivations \mathbf{z} , which provides a lower bound.

Learning can be performed by maximizing the log marginal likelihood,

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^N \log p(\mathbf{y}^{(i)} | \mathbf{w}^{(i)}; \boldsymbol{\theta}) \quad [12.25]$$

$$= \sum_{i=1}^N \log \sum_{\mathbf{z}} p(\mathbf{y}^{(i)}, \mathbf{z}^{(i)} | \mathbf{w}^{(i)}; \boldsymbol{\theta}). \quad [12.26]$$

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,

$$\frac{\partial \ell_i}{\partial \boldsymbol{\theta}} = \sum_{\mathbf{z}} p(\mathbf{z} | \mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) \mathbf{f}(\mathbf{w}, \mathbf{z}, \mathbf{y}) - \sum_{\mathbf{y}', \mathbf{z}'} p(\mathbf{y}', \mathbf{z}' | \mathbf{w}; \boldsymbol{\theta}) \mathbf{f}(\mathbf{w}, \mathbf{z}', \mathbf{y}') \quad [12.27]$$

$$= E_{\mathbf{z}|\mathbf{y}, \mathbf{w}} \mathbf{f}(\mathbf{w}, \mathbf{z}, \mathbf{y}) - E_{\mathbf{y}, \mathbf{z}|\mathbf{w}} \mathbf{f}(\mathbf{w}, \mathbf{z}, \mathbf{y}) \quad [12.28]$$

Algorithm 16 Latent variable perceptron

```

1: procedure LatentVariablePerceptron( $\mathbf{w}^{(1:N)}, \mathbf{y}^{(1:N)}$ )
2:    $\boldsymbol{\theta} \leftarrow \mathbf{0}$ 
3:   repeat
4:     Select an instance  $i$ 
5:      $\mathbf{z}^{(i)} \leftarrow \text{argmax}_{\mathbf{z}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{z}, \mathbf{y}^{(i)})$ 
6:      $\hat{\mathbf{y}}, \hat{\mathbf{z}} \leftarrow \text{argmax}_{\mathbf{y}', \mathbf{z}'} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{z}', \mathbf{y}')$ 
7:      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{z}^{(i)}, \mathbf{y}^{(i)}) - \mathbf{f}(\mathbf{w}^{(i)}, \hat{\mathbf{z}}, \hat{\mathbf{y}})$ 
8:   until tired
9:   return  $\boldsymbol{\theta}$ 

```

6568 Both expectations can be computed via bottom-up algorithms like inside-outside. Alternatively,
 6569 we can again maximize rather than marginalize over derivations for an approximate solution.
 6570 In either case, the first term of the gradient requires us to identify derivations \mathbf{z} that are
 6571 compatible with the logical form \mathbf{y} . This can be done in a bottom-up dynamic programming
 6572 algorithm, by having each cell in the table $t[i, j, X]$ include the set of all possible logical
 6573 forms for $X \rightsquigarrow \mathbf{w}_{i+1:j}$. The resulting table may therefore be much larger than in syntactic
 6574 parsing. This can be controlled by using pruning to eliminate intermediate analyses that
 6575 are incompatible with the final logical form \mathbf{y} (Zettlemoyer and Collins, 2005), or by using
 6576 beam search and restricting the size of each cell to some fixed constant (Liang et al., 2013).

6577 If we replace each expectation in Equation 12.28 with argmax and then apply stochastic
 6578 gradient descent to learn the weights, we obtain the latent variable perceptron, a simple
 6579 and general algorithm for learning with missing data. The algorithm is shown in its most
 6580 basic form in Algorithm 16, but the usual tricks such as averaging and margin loss can
 6581 be applied (Yu and Joachims, 2009). Aside from semantic parsing, the latent variable
 6582 perceptron has been used in tasks such as machine translation (Liang et al., 2006) and
 6583 named entity recognition (Sun et al., 2009). In latent conditional random fields, we use the
 6584 full expectations rather than maximizing over the hidden variable. This model has also been
 6585 employed in a range of problems beyond semantic parsing, including parse reranking (Koo
 6586 and Collins, 2005) and gesture recognition (Quattoni et al., 2007).

6587 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}

6588 Similarly, in a robotic control setting, the denotation of a command would be an action or
 6589 sequence of actions (Artzi and Zettlemoyer, 2013). In both cases, the idea is to reward the
 6590 semantic parser for choosing an analysis whose denotation is correct: the right answer to
 6591 the question, or the right action.

Learning from logical forms was made possible by summing or maxing over derivations. This idea can be carried one step further, summing or maxing over all logical forms with the correct denotation. Let $v_i(\mathbf{y}) \in \{0, 1\}$ be a validation function, which assigns a binary score indicating whether the denotation $[\![\mathbf{y}]\!]$ for the text $\mathbf{w}^{(i)}$ is correct. We can then learn by maximizing a conditional-likelihood objective,

$$\ell^{(i)}(\boldsymbol{\theta}) = \log \sum_{\mathbf{y}} v_i(\mathbf{y}) \times p(\mathbf{y} \mid \mathbf{w}; \boldsymbol{\theta}) \quad [12.29]$$

$$= \log \sum_{\mathbf{y}} v_i(\mathbf{y}) \times \sum_{\mathbf{z}} p(\mathbf{y}, \mathbf{z} \mid \mathbf{w}; \boldsymbol{\theta}), \quad [12.30]$$

6592 which sums over all derivations \mathbf{z} of all valid logical forms, $\{\mathbf{y} : v_i(\mathbf{y}) = 1\}$. This
 6593 corresponds to the log-probability that the semantic parser produces a logical form with a
 6594 valid denotation.

Differentiating with respect to $\boldsymbol{\theta}$, we obtain,

$$\frac{\partial \ell^{(i)}}{\partial \boldsymbol{\theta}} = \sum_{\mathbf{y}, \mathbf{z}: v_i(\mathbf{y})=1} p(\mathbf{y}, \mathbf{z} \mid \mathbf{w}) \mathbf{f}(\mathbf{w}, \mathbf{z}, \mathbf{y}) - \sum_{\mathbf{y}', \mathbf{z}'} p(\mathbf{y}', \mathbf{z}' \mid \mathbf{w}) \mathbf{f}(\mathbf{w}, \mathbf{z}', \mathbf{y}'), \quad [12.31]$$

6595 which is the usual difference in feature expectations. The positive term computes the
 6596 expected feature expectations conditioned on the denotation being valid, while the second
 6597 term computes the expected feature expectations according to the current model, without
 6598 regard to the ground truth. Large-margin learning formulations are also possible for this
 6599 problem. For example, Artzi and Zettlemoyer (2013) generate a set of valid and invalid
 6600 derivations, and then impose a constraint that all valid derivations should score higher
 6601 than all invalid derivations. This constraint drives a perceptron-like learning rule.

6602 Additional resources

6603 A key issue not considered here is how to handle semantic underspecification: cases in which
 6604 there are multiple semantic interpretations for a single syntactic structure. Quantifier

6605 scope ambiguity is a classic example. Blackburn and Bos (2005) enumerate a number
 6606 of approaches to this issue, and also provide links between natural language semantics
 6607 and computational inference techniques. Much of the contemporary research on semantic
 6608 parsing uses the framework of combinatory categorial grammar (CCG). Carpenter (1997)
 6609 provides a comprehensive treatment of how CCG can support compositional semantic
 6610 analysis. Another recent area of research is the semantics of multi-sentence texts. This can
 6611 be handled with models of dynamic semantics, such as dynamic predicate logic (Groenendijk
 6612 and Stokhof, 1991).

6613 Alternative readings on formal semantics include an “informal” reading from Levy and
 6614 Manning (2009), and a more involved introduction from Briscoe (2011). To learn more
 6615 about ongoing research on data-driven semantic parsing, readers may consult the survey
 6616 article by Liang and Potts (2015), tutorial slides and videos by Artzi and Zettlemoyer
 6617 (2013),¹² and the source code by Yoav Artzi¹³ and Percy Liang.¹⁴

6618 Exercises

- 6619 1. Derive the modus ponens inference rule, which states that if we know $\phi \Rightarrow \psi$ and ϕ ,
 6620 then ψ must be true. The derivation can be performed using the definition of the
 6621 \Rightarrow operator and some of the laws provided in § 12.2.1, plus one additional identity:
 6622 $\perp \vee \phi = \phi$.
- 6623 2. Convert the following examples into first-order logic, using the relations can-sleep,
 6624 makes-noise, and brother.
 - 6625 • If Abigail makes noise, no one can sleep.
 - 6626 • If Abigail makes noise, someone cannot sleep.
 - 6627 • None of Abigail’s brothers can sleep.
 - 6628 • If one of Abigail’s brothers makes noise, Abigail cannot sleep.
- 6629 3. Extend the grammar fragment G_1 to include the ditransitive verb teaches and the
 6630 proper noun Swahili. Show how to derive the interpretation for the sentence Alex
 6631 teaches Brit Swahili, which should be $\text{teaches}(\text{alex}, \text{brit}, \text{swahili})$. The grammar need
 6632 not be in Chomsky Normal Form. For the ditransitive verb, use NP_1 and NP_2 to
 6633 indicate the two direct objects.
- 6634 4. Derive the semantic interpretation for the sentence Alex likes every dog, using grammar
 6635 fragment G_2 .

¹²Videos are currently available at <http://yoavartzi.com/tutorial/>

¹³<http://yoavartzi.com/spf>

¹⁴<https://github.com/percyliang/sempre>

- 6636 5. Extend the grammar fragment G_2 to handle adjectives, so that the meaning of an
 6637 angry dog is $\lambda P. \exists x \text{dog}(x) \wedge \text{angry}(x) \wedge P(x)$. Specifically, you should supply the
 6638 lexical entry for the adjective angry, and you should specify the syntactic-semantic
 6639 productions $\text{NP} \rightarrow \text{Det NOM}$, $\text{NOM} \rightarrow \text{Jj NOM}$, and $\text{NOM} \rightarrow \text{Nn}$.
- 6640 6. Extend your answer to the previous question to cover copula constructions with
 6641 predicative adjectives, such as Alex is angry. The interpretation should be $\text{angry}(\text{alex})$.
 6642 You should add a verb phrase production $\text{VP} \rightarrow \text{V}_{\text{cop}} \text{ Jj}$, and a terminal production
 6643 $\text{V}_{\text{cop}} \rightarrow \text{is}$. Show why your grammar extensions result in the correct interpretation.
- 6644 7. In Figure 12.6 and Figure 12.7, we treat the plurals shoots and leaves as entities.
 6645 Revise G_2 so that the interpretation of Alex eats leaves is $\forall x. (\text{leaf}(x) \Rightarrow \text{eats}(\text{alex}, x))$,
 6646 and show the resulting perceptron update.
- 6647 8. Statements like every student eats a pizza have two possible interpretations, depending
 6648 on quantifier scope:

$$\forall x \exists y \text{pizza}(y) \wedge (\text{student}(x) \Rightarrow \text{eats}(x, y)) \quad [12.32]$$

$$\exists y \forall x \text{pizza}(y) \wedge (\text{student}(x) \Rightarrow \text{eats}(x, y)) \quad [12.33]$$

6647 Explain why these interpretations really are different, and modify the grammar G_2
 6648 so that it can produce both interpretations.

- 6649 9. Derive Equation 12.27.
- 6650 10. In the GeoQuery domain, give a natural language query that has multiple plausible
 6651 semantic interpretations with the same denotation. List both interpretations and the
 6652 denotation.

6653 Hint: There are many ways to do this, but one approach involves using toponyms
 6654 (place names) that could plausibly map to several different entities in the model.

6655 Chapter 13

6656 Predicate-argument semantics

6657 This chapter considers more “lightweight” semantic representations, which discard some
6658 aspects of first-order logic, but focus on predicate-argument structures. Let’s begin by
6659 thinking about the semantics of events, with a simple example:

6660 (13.1) Asha gives Boyang a book.

6661 A first-order logical representation of this sentence is,

$$\exists x. \text{book}(x) \wedge \text{give}(\text{Asha}, \text{Boyang}, x) \quad [13.1]$$

6662 In this representation, we define variable x for the book, and we link the strings Asha
6663 and Boyang to entities Asha and Boyang. Because the action of giving involves a giver, a
6664 recipient, and a gift, the predicate give must take three arguments.

6665 Now suppose we have additional information about the event:

6666 (13.2) Yesterday, Asha reluctantly gave Boyang a book.

6667 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 ,

$$\begin{aligned} \exists e, x. & \text{give-event}(e) \wedge \text{giver}(e, \text{Asha}) \wedge \text{gift}(e, x) \wedge \text{book}(e, x) \wedge \text{recipient}(e, \text{Boyang}) \\ & \wedge \text{time}(e, \text{Yesterday}) \wedge \text{manner}(e, \text{reluctantly}) \end{aligned}$$

6668 In this way, each argument of the event — the giver, the recipient, the gift — can be
 6669 represented with a relation of its own, linking the argument to the event e . The expression
 6670 $\text{giver}(e, \text{Asha})$ says that Asha plays the role of giver in the event. This reformulation handles
 6671 the problem of optional information such as the time or manner of the event, which are
 6672 called adjuncts. Unlike arguments, adjuncts are not a mandatory part of the relation, but
 6673 under this representation, they can be expressed with additional logical relations that are
 6674 conjoined to the semantic interpretation of the sentence.¹

6675 The event semantic representation can be applied to nested clauses, e.g.,

6676 (13.3) Chris sees Asha pay Boyang.

This is done by using the event variable as an argument:

$$\begin{aligned} \exists e_1 \exists e_2 & \text{See-Event}(e_1) \wedge \text{seer}(e_1, \text{Chris}) \wedge \text{sight}(e_1, e_2) \\ & \wedge \text{Pay-Event}(e_2) \wedge \text{payer}(e_2, \text{Asha}) \wedge \text{payee}(e_2, \text{Boyang}) \end{aligned} \quad [13.3]$$

6677 As with first-order logic, the goal of event semantics is to provide a representation that
 6678 generalizes over many surface forms. Consider the following paraphrases of (13.1):

- 6679 (13.4) Asha gives a book to Boyang.
- 6680 (13.5) A book is given to Boyang by Asha.
- 6681 (13.6) A book is given by Asha to Boyang.
- 6682 (13.7) The gift of a book from Asha to Boyang ...

6683 All have the same event semantic meaning as Equation 13.1, but the ways in which the
 6684 meaning can be expressed are diverse. The final example does not even include a verb:
 6685 events are often introduced by verbs, but as shown by (13.7), the noun gift can introduce
 6686 the same predicate, with the same accompanying arguments.

6687 Semantic role labeling (SRL) is a relaxed form of semantic parsing, in which each
 6688 semantic role is filled by a set of tokens from the text itself. This is sometimes called
 6689 “shallow semantics” because, unlike model-theoretic semantic parsing, role fillers need not
 6690 be symbolic expressions with denotations in some world model. A semantic role labeling
 6691 system is required to identify all predicates, and then specify the spans of text that fill
 6692 each role. To give a sense of the task, here is a more complicated example:

- 6693 (13.8) Boyang wants Asha to give him a linguistics book.

¹This representation is often called Neo-Davidsonian event semantics. The use of existentially-quantified event variables was proposed by Davidson (1967) to handle the issue of optional adjuncts. In Neo-Davidsonian semantics, this treatment of adjuncts is extended to mandatory arguments as well (e.g., Parsons, 1990).

6694 In this example, there are two predicates, expressed by the verbs want and give. Thus, a
 6695 semantic role labeler might return the following output:

- 6696 • (Predicate : wants, Wanter : Boyang, Desire : Asha to give him a linguistics book)
 6697 • (Predicate : give, Giver : Asha, Recipient : him, Gift : a linguistics book)

6698 Boyang and him may refer to the same person, but the semantic role labeling is not required
 6699 to resolve this reference. Other predicate-argument representations, such as Abstract
 6700 Meaning Representation (AMR), do require reference resolution. We will return to AMR
 6701 in § 13.3, but first, let us further consider the definition of semantic roles.

6702 **13.1 Semantic roles**

6703 In event semantics, it is necessary to specify a number of additional logical relations to link
 6704 arguments to events: giver, recipient, seer, sight, etc. Indeed, every predicate requires a set
 6705 of logical relations to express its own arguments. In contrast, adjuncts such as time and
 6706 manner are shared across many types of events. A natural question is whether it is possible
 6707 to treat mandatory arguments more like adjuncts, by identifying a set of generic argument
 6708 types that are shared across many event predicates. This can be further motivated by
 6709 examples involving related verbs:

- 6710 (13.9) Asha gave Boyang a book.
 6711 (13.10) Asha loaned Boyang a book.
 6712 (13.11) Asha taught Boyang a lesson.
 6713 (13.12) Asha gave Boyang a lesson.

6714 The respective roles of Asha, Boyang, and the book are nearly identical across the first two
 6715 examples. The third example is slightly different, but the fourth example shows that the
 6716 roles of giver and teacher can be viewed as related.

6717 One way to think about the relationship between roles such as giver and teacher is by
 6718 enumerating the set of properties that an entity typically possesses when it fulfills these
 6719 roles: givers and teachers are usually animate (they are alive and sentient) and volitional
 6720 (they choose to enter into the action).² In contrast, the thing that gets loaned or taught
 6721 is usually not animate or volitional; furthermore, it is unchanged by the event.

6722 Building on these ideas, thematic roles generalize across predicates by leveraging the
 6723 shared semantic properties of typical role fillers (Fillmore, 1968). For example, in examples

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.

	Asha	gave	Boyang	a book
VerbNet	Agent		Recipient	Theme
PropBank	Arg0: giver		Arg2: entity given to	Arg1: thing given
FrameNet	Donor		Recipient	Theme
	Asha	taught	Boyang	algebra
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

6724 (13.9-13.12), Asha plays a similar role in all four sentences, which we will call the agent.
 6725 This reflects several shared semantic properties: she is the one who is actively and intentionally
 6726 performing the action, while Boyang is a more passive participant; the book and the lesson
 6727 would play a different role, as non-animate participants in the event.

6728 Example annotations from three well known systems are shown in Figure 13.1. We will
 6729 now discuss these systems in more detail.

6730 13.1.1 VerbNet

6731 VerbNet (Kipper-Schuler, 2005) is a lexicon of verbs, and it includes thirty “core” thematic
 6732 roles played by arguments to these verbs. Here are some example roles, accompanied by
 6733 their definitions from the VerbNet Guidelines.³

- 6734 • Agent: “Actor in an event who initiates and carries out the event intentionally or
 6735 consciously, and who exists independently of the event.”
- 6736 • Patient: “Undergoer in an event that experiences a change of state, location or
 6737 condition, that is causally involved or directly affected by other participants, and
 6738 exists independently of the event.”
- 6739 • Recipient: “Destination that is animate”
- 6740 • Theme: “Undergoer that is central to an event or state that does not have control
 6741 over the way the event occurs, is not structurally changed by the event, and/or is
 6742 characterized as being in a certain position or condition throughout the state.”
- 6743 • Topic: “Theme characterized by information content transferred to another participant.”

³http://verbs.colorado.edu/verb-index/VerbNet_Guidelines.pdf

6744 VerbNet roles are organized in a hierarchy, so that a topic is a type of theme, which in
6745 turn is a type of undergoer, which is a type of participant, the top-level category.

6746 In addition, VerbNet organizes verb senses into a class hierarchy, in which verb senses
6747 that have similar meanings are grouped together. Recall from § 4.2 that multiple meanings
6748 of the same word are called senses, and that WordNet identifies senses for many English
6749 words. VerbNet builds on WordNet, so that verb classes are identified by the WordNet
6750 senses of the verbs that they contain. For example, the verb class give-13.1 includes the
6751 first WordNet sense of loan and the second WordNet sense of lend.

6752 Each VerbNet class or subclass takes a set of thematic roles. For example, give-13.1
6753 takes arguments with the thematic roles of Agent, Theme, and Recipient;⁴ the predicate
6754 teach takes arguments with the thematic roles Agent, Topic, Recipient, and Source.⁵ So
6755 according to VerbNet, Asha and Boyang play the roles of Agent and Recipient in the
6756 sentences,

6757 (13.13) Asha gave Boyang a book.

6758 (13.14) Asha taught Boyang algebra.

6759 The book and algebra are both Themes, but algebra is a subcategory of Theme — a Topic
6760 — because it consists of information content that is given to the receiver.

6761 13.1.2 Proto-roles and PropBank

6762 Detailed thematic role inventories of the sort used in VerbNet are not universally accepted.
6763 For example, Dowty (1991, pp. 547) notes that “Linguists have often found it hard to
6764 agree on, and to motivate, the location of the boundary between role types.” He argues
6765 that a solid distinction can be identified between just two proto-roles:

6766 Proto-Agent. Characterized by volitional involvement in the event or state; sentience
6767 and/or perception; causing an event or change of state in another participant; movement;
6768 exists independently of the event.

6769 Proto-Patient. Undergoes change of state; causally affected by another participant; stationary
6770 relative to the movement of another participant; does not exist independently of the
6771 event.⁶

⁴<https://verbs.colorado.edu/verb-index/vn/give-13.1.php>

⁵https://verbs.colorado.edu/verb-index/vn/transfer_mesg-37.1.1.php

⁶Reisinger et al. (2015) ask crowd workers to annotate these properties directly, finding that annotators tend to agree on the properties of each argument. They also find that in English, arguments having more proto-agent properties tend to appear in subject position, while arguments with more proto-patient properties appear in object position.

6772 In the examples in Figure 13.1, Asha has most of the proto-agent properties: in giving
 6773 the book to Boyang, she is acting volitionally (as opposed to Boyang got a book from
 6774 Asha, in which it is not clear whether Asha gave up the book willingly); she is sentient; she
 6775 causes a change of state in Boyang; she exists independently of the event. Boyang has some
 6776 proto-agent properties: he is sentient and exists independently of the event. But he also
 6777 some proto-patient properties: he is the one who is causally affected and who undergoes
 6778 change of state. The book that Asha gives Boyang has even fewer of the proto-agent
 6779 properties: it is not volitional or sentient, and it has no causal role. But it also lacks many
 6780 of the proto-patient properties: it does not undergo change of state, exists independently
 6781 of the event, and is not stationary.

6782 The Proposition Bank, or PropBank (Palmer et al., 2005), builds on this basic agent-patient
 6783 distinction, as a middle ground between generic thematic roles and roles that are specific
 6784 to each predicate. Each verb is linked to a list of numbered arguments, with Arg0
 6785 as the proto-agent and Arg1 as the proto-patient. Additional numbered arguments are
 6786 verb-specific. For example, for the predicate teach,⁷ the arguments are:

- 6787 • Arg0: the teacher
- 6788 • Arg1: the subject
- 6789 • Arg2: the student(s)

6790 Verbs may have any number of arguments: for example, want and get have five, while
 6791 eat has only Arg0 and Arg1. In addition to the semantic arguments found in the frame
 6792 files, roughly a dozen general-purpose adjuncts may be used in combination with any verb.
 6793 These are shown in Table 13.1.

6794 PropBank-style semantic role labeling is annotated over the entire Penn Treebank. This
 6795 annotation includes the sense of each verbal predicate, as well as the argument spans.

6796 13.1.3 FrameNet

6797 Semantic frames are descriptions of situations or events. Frames may be evoked by one
 6798 of their lexical units (often a verb, but not always), and they include some number of
 6799 frame elements, which are like roles (Fillmore, 1976). For example, the act of teaching
 6800 is a frame, and can be evoked by the verb taught; the associated frame elements include
 6801 the teacher, the student(s), and the subject being taught. Frame semantics has played a
 6802 significant role in the history of artificial intelligence, in the work of Minsky (1974) and
 6803 Schank and Abelson (1977). In natural language processing, the theory of frame semantics
 6804 has been implemented in FrameNet (Fillmore and Baker, 2009), which consists of a lexicon

⁷<http://verbs.colorado.edu/propbank/framesets-english-aliases/teach.html>

Tmp	time	Boyang ate a bagel [Am-Tmp yesterday].
Loc	location	Asha studies in [Am-Loc Stuttgart]
Mod	modal verb	Asha [Am-Mod will] study in Stuttgart
Adv	general purpose	[Am-Adv Luckily], Asha knew algebra.
Mnr	manner	Asha ate [Am-Mnr aggressively].
Dis	discourse connective	[Am-Dis However], Asha prefers algebra.
Prp	purpose	Barry studied [Am-Prp to pass the bar].
Dir	direction	Workers dumped burlap sacks [Am-Dir into a bin].
Neg	negation	Asha does [Am-Neg not] speak Albanian.
Ext	extent	Prices increased [Am-Ext 4%].
Cau	cause	Boyang returned the book [Am-Cau because it was overdue].

Table 13.1: PropBank adjuncts (Palmer et al., 2005), sorted by frequency in the corpus

6805 of roughly 1000 frames, and a corpus of more than 200,000 “exemplar sentences,” in which
 6806 the frames and their elements are annotated.⁸

6807 Rather than seeking to link semantic roles such as teacher and giver into thematic roles
 6808 such as agent, FrameNet aggressively groups verbs into frames, and links semantically-related
 6809 roles across frames. For example, the following two sentences would be annotated identically
 6810 in FrameNet:

6811 (13.15) Asha taught Boyang algebra.

6812 (13.16) Boyang learned algebra from Asha.

6813 This is because teach and learn are both lexical units in the education_teaching frame.
 6814 Furthermore, roles can be shared even when the frames are distinct, as in the following
 6815 two examples:

6816 (13.17) Asha gave Boyang a book.

6817 (13.18) Boyang got a book from Asha.

6818 The giving and getting frames both have Recipient and Theme elements, so Boyang and the
 6819 book would play the same role. Asha’s role is different: she is the Donor in the giving frame,
 6820 and the Source in the getting frame. FrameNet makes extensive use of multiple inheritance
 6821 to share information across frames and frame elements: for example, the commerce_sell
 6822 and lending frames inherit from giving frame.

⁸Current details and data can be found at <https://framenet.icsi.berkeley.edu/>

6823 13.2 Semantic role labeling

6824 The task of semantic role labeling is to identify the parts of the sentence comprising the
 6825 semantic roles. In English, this task is typically performed on the PropBank corpus, with
 6826 the goal of producing outputs in the following form:

6827 (13.19) [Arg0 Asha] [give.01 gave] [Arg2 Boyang’s mom] [Arg1 a book] [AM-TMP yesterday].

6828 Note that a single sentence may have multiple verbs, and therefore a given word may be
 6829 part of multiple role-fillers:

6830 (13.20) [Arg0 Asha] [want.01 wanted]
 Asha wanted
 6831 [Arg1 Boyang to give her the book].
 [Arg0 Boyang] [give.01 to give] [Arg2 her] [Arg1 the book].

6832 13.2.1 Semantic role labeling as classification

6833 PropBank is annotated on the Penn Treebank, and annotators used phrasal constituents
 6834 (\S 9.2.2) to fill the roles. PropBank semantic role labeling can be viewed as the task of
 6835 assigning to each phrase a label from the set $\mathcal{R} = \{\emptyset, \text{Pred}, \text{Arg0}, \text{Arg1}, \text{Arg2}, \dots, \text{Am-Loc}, \text{Am-Tmp}, \dots\}$
 6836 with respect to each predicate. If we treat semantic role labeling as a classification problem,
 6837 we obtain the following functional form:

$$\hat{y}_{(i,j)} = \underset{y}{\operatorname{argmax}} \psi(\mathbf{w}, y, i, j, \rho, \tau), \quad [13.4]$$

6838 where,

- 6839 • (i, j) indicates the span of a phrasal constituent $(w_{i+1}, w_{i+2}, \dots, w_j)$,⁹
- 6840 • \mathbf{w} represents the sentence as a sequence of tokens;
- 6841 • ρ is the index of the predicate verb in \mathbf{w} ;
- 6842 • τ is the structure of the phrasal constituent parse of \mathbf{w} .

6843 Early work on semantic role labeling focused on discriminative feature-based models,
 6844 where $\psi(\mathbf{w}, y, i, j, \rho, \tau) = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, y, i, j, \rho, \tau)$. Table 13.2 shows the features used in a
 6845 seminal paper on FrameNet semantic role labeling (Gildea and Jurafsky, 2002). By 2005

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 taught would be VP → VBD NP PP.

Table 13.2: Features used in semantic role labeling by Gildea and Jurafsky (2002).

6846 there were several systems for PropBank semantic role labeling, and their approaches and
 6847 feature sets are summarized by Carreras and Màrquez (2005). Typical features include:
 6848 the phrase type, head word, part-of-speech, boundaries, and neighbors of the proposed
 6849 argument $w_{i+1:j}$; the word, lemma, part-of-speech, and voice of the verb w_ρ (active or
 6850 passive), as well as features relating to its frameset; the distance and path between the
 6851 verb and the proposed argument. In this way, semantic role labeling systems are high-level
 6852 “consumers” in the NLP stack, using features produced from lower-level components such
 6853 as part-of-speech taggers and parsers. More comprehensive feature sets are enumerated by
 6854 Das et al. (2014) and Täckström et al. (2015).

6855 A particularly powerful class of features relate to the syntactic path between the
 6856 argument and the predicate. These features capture the sequence of moves required to get
 6857 from the argument to the verb by traversing the phrasal constituent parse of the sentence.
 6858 The idea of these features is to capture syntactic regularities in how various arguments are
 6859 realized. Syntactic path features are best illustrated by example, using the parse tree in
 6860 Figure 13.2:

- 6861 • The path from Asha to the verb taught is Nnp↑NP↑S↓VP↓Vbd. The first part of
 6862 the path, NNP↑NP↑S, means that we must travel up the parse tree from the Nnp
 6863 tag (proper noun) to the S (sentence) constituent. The second part of the path,
 6864 S↓VP↓Vbd, means that we reach the verb by producing a VP (verb phrase) from

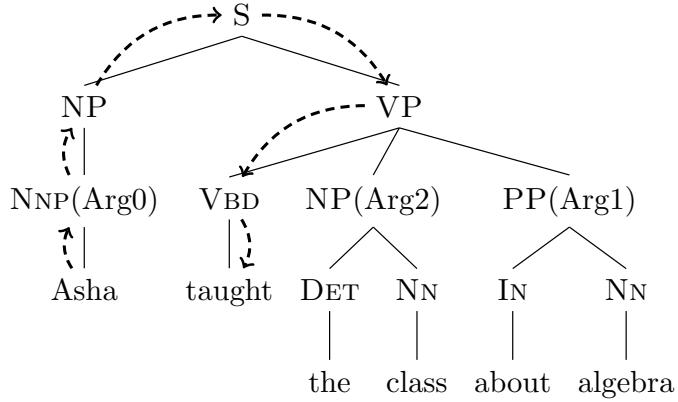


Figure 13.2: Semantic role labeling on the phrase-structure parse tree for a sentence. The dashed line indicates the syntactic path from Asha to the predicate verb taught.

6865 the S constituent, and then by producing a Vbd (past tense verb). This feature is
 6866 consistent with Asha being in subject position, since the path includes the sentence
 6867 root S.

- 6868 • The path from the class to taught is $\text{NP} \uparrow \text{VP} \downarrow \text{Vbd}$. This is consistent with the class
 6869 being in object position, since the path passes through the VP node that dominates
 6870 the verb taught.

6871 Because there are many possible path features, it can also be helpful to look at smaller
 6872 parts: for example, the upward and downward parts can be treated as separate features;
 6873 another feature might consider whether S appears anywhere in the path.

6874 Rather than using the constituent parse, it is also possible to build features from
 6875 the dependency path between the head word of each argument and the verb (Pradhan
 6876 et al., 2005). Using the Universal Dependency part-of-speech tagset and dependency
 6877 relations (Nivre et al., 2016), the dependency path from Asha to taught is $\text{Propn} \xleftarrow[\text{Nsubj}]{} \text{Verb}$,
 6878 because taught is the head of a relation of type $\xleftarrow[\text{Nsubj}]{} \text{Verb}$ with Asha. Similarly, the dependency
 6879 path from class to taught is $\text{Noun} \xleftarrow[\text{dobj}]{} \text{Verb}$, because class heads the noun phrase that is
 6880 a direct object of taught. A more interesting example is Asha wanted to teach the class,
 6881 where the path from Asha to teach is $\text{Propn} \xleftarrow[\text{Nsubj}]{} \text{Verb} \rightarrow[\text{Xcomp}] \text{Verb}$. The right-facing arrow in
 6882 second relation indicates that wanted is the head of its Xcomp relation with teach.

6883 13.2.2 Semantic role labeling as constrained optimization

6884 A potential problem with treating SRL as a classification problem is that there are a
 6885 number of sentence-level constraints, which a classifier might violate.

- 6886 • For a given verb, there can be only one argument of each type (Arg0, Arg1, etc.)
- 6887 • Arguments cannot overlap. This problem arises when we are labeling the phrases in
 6888 a constituent parse tree, as shown in Figure 13.2: if we label the PP about algebra
 6889 as an argument or adjunct, then its children about and algebra must be labeled as
 6900 \emptyset . The same constraint also applies to the syntactic ancestors of this phrase.

6891 These constraints introduce dependencies across labeling decisions. In structure prediction
 6892 problems such as sequence labeling and parsing, such dependencies are usually handled by
 6893 defining a scoring over the entire structure, \mathbf{y} . Efficient inference requires that the global
 6894 score decomposes into local parts: for example, in sequence labeling, the scoring function
 6895 decomposes into scores of pairs of adjacent tags, permitting the application of the Viterbi
 6896 algorithm for inference. But the constraints that arise in semantic role labeling are less
 6897 amenable to local decomposition.¹⁰ We therefore consider constrained optimization as an
 6898 alternative solution.

Let the set $\mathcal{C}(\tau)$ refer to all labelings that obey the constraints introduced by the parse τ . The semantic role labeling problem can be reformulated as a constrained optimization over $\mathbf{y} \in \mathcal{C}(\tau)$,

$$\begin{aligned} \max_{\mathbf{y}} \quad & \sum_{(i,j) \in \tau} \psi(\mathbf{w}, y_{i,j}, i, j, \rho, \tau) \\ \text{s.t.} \quad & \mathbf{y} \in \mathcal{C}(\tau). \end{aligned} \quad [13.5]$$

6899 In this formulation, the objective (shown on the first line) is a separable function of each
 6900 individual labeling decision, but the constraints (shown on the second line) apply to the
 6901 overall labeling. The sum $\sum_{(i,j) \in \tau}$ indicates that we are summing over all constituent spans
 6902 in the parse τ . The expression s.t. in the second line means that we maximize the objective
 6903 subject to the constraint $\mathbf{y} \in \mathcal{C}(\tau)$.

6904 A number of practical algorithms exist for restricted forms of constrained optimization.
 6905 One such restricted form is integer linear programming, in which the objective and constraints
 6906 are linear functions of integer variables. To formulate SRL as an integer linear program,
 6907 we begin by rewriting the labels as a set of binary variables $\mathbf{z} = \{z_{i,j,r}\}$ (Punyakanok et al.,
 6908 2008),

$$z_{i,j,r} = \begin{cases} 1, & y_{i,j} = r \\ 0, & \text{otherwise,} \end{cases} \quad [13.6]$$

¹⁰Dynamic programming solutions have been proposed by Tromble and Eisner (2006) and Täckström et al. (2015), but they involve creating a trellis structure whose size is exponential in the number of labels.

6909 where $r \in \mathcal{R}$ is a label in the set $\{\text{Arg0}, \text{Arg1}, \dots, \text{Am-Loc}, \dots, \emptyset\}$. Thus, the variables \mathbf{z}
 6910 are a binarized version of the semantic role labeling \mathbf{y} .

The objective can then be formulated as a linear function of \mathbf{z} .

$$\sum_{(i,j) \in \tau} \psi(\mathbf{w}, y_{i,j}, i, j, \rho, \tau) = \sum_{i,j,r} \psi(\mathbf{w}, r, i, j, \rho, \tau) \times z_{i,j,r}, \quad [13.7]$$

6911 which is the sum of the scores of all relations, as indicated by $z_{i,j,r}$.

Constraints Integer linear programming permits linear inequality constraints, of the general form $\mathbf{A}\mathbf{z} \leq \mathbf{b}$, where the parameters \mathbf{A} and \mathbf{b} define the constraints. To make this more concrete, let's start with the constraint that each non-null role type can occur only once in a sentence. This constraint can be written,

$$\forall r \neq \emptyset, \quad \sum_{(i,j) \in \tau} z_{i,j,r} \leq 1. \quad [13.8]$$

6912 Recall that $z_{i,j,r} = 1$ iff the span (i, j) has label r ; this constraint says that for each possible
 6913 label $r \neq \emptyset$, there can be at most one (i, j) such that $z_{i,j,r} = 1$. Rewriting this constraint
 6914 can be written in the form $\mathbf{A}\mathbf{z} \leq \mathbf{b}$, as you will find if you complete the exercises at the
 6915 end of the chapter.

Now consider the constraint that labels cannot overlap. Let's define the convenience function $o((i, j), (i', j')) = 1$ iff (i, j) overlaps (i', j') , and zero otherwise. Thus, o will indicate if a constituent (i', j') is either an ancestor or descendant of (i, j) . The constraint is that if two constituents overlap, only one can have a non-null label:

$$\forall (i, j) \in \tau, \quad \sum_{(i', j') \in \tau} \sum_{r \neq \emptyset} o((i, j), (i', j')) \times z_{i',j',r} \leq 1, \quad [13.9]$$

6916 where $o((i, j), (i, j)) = 1$.

In summary, the semantic role labeling problem can thus be rewritten as the following integer linear program,

$$\max_{\mathbf{z} \in \{0,1\}^{|\tau|}} \quad \sum_{(i,j) \in \tau} \sum_{r \in \mathcal{R}} z_{i,j,r} \psi_{i,j,r} \quad [13.10]$$

$$s.t. \quad \forall r \neq \emptyset, \quad \sum_{(i,j) \in \tau} z_{i,j,r} \leq 1. \quad [13.11]$$

$$\forall (i, j) \in \tau, \quad \sum_{(i', j') \in \tau} \sum_{r \neq \emptyset} o((i, j), (i', j')) \times z_{i',j',r} \leq 1. \quad [13.12]$$

6917 Learning with constraints Learning can be performed in the context of constrained optimization
 6918 using the usual perceptron or large-margin classification updates. Because constrained
 6919 inference is generally more time-consuming, a key question is whether it is necessary to
 6920 apply the constraints during learning. Chang et al. (2008) find that better performance
 6921 can be obtained by learning without constraints, and then applying constraints only when
 6922 using the trained model to predict semantic roles for unseen data.

6923 How important are the constraints? Das et al. (2014) find that an unconstrained, classification-based
 6924 method performs nearly as well as constrained optimization for FrameNet parsing: while
 6925 it commits many violations of the “no-overlap” constraint, the overall F_1 score is less
 6926 than one point worse than the score at the constrained optimum. Similar results were
 6927 obtained for PropBank semantic role labeling by Punyakanok et al. (2008). He et al.
 6928 (2017) find that constrained inference makes a bigger impact if the constraints are based
 6929 on manually-labeled “gold” syntactic parses. This implies that errors from the syntactic
 6930 parser may limit the effectiveness of the constraints. Punyakanok et al. (2008) hedge against
 6931 parser error by including constituents from several different parsers; any constituent can
 6932 be selected from any parse, and additional constraints ensure that overlapping constituents
 6933 are not selected.

6934 Implementation Integer linear programming solvers such as glpk,¹¹ cplex,¹² and Gurobi¹³
 6935 allow inequality constraints to be expressed directly in the problem definition, rather
 6936 than in the matrix form $\mathbf{A}\mathbf{z} \leq \mathbf{b}$. The time complexity of integer linear programming is
 6937 theoretically exponential in the number of variables $|\mathbf{z}|$, but in practice these off-the-shelf
 6938 solvers obtain good solutions efficiently. Das et al. (2014) report that the cplex solver
 6939 requires 43 seconds to perform inference on the FrameNet test set, which contains 4,458
 6940 predicates.

6941 Recent work has shown that many constrained optimization problems in natural language
 6942 processing can be solved in a highly parallelized fashion, using optimization techniques such
 6943 as dual decomposition, which are capable of exploiting the underlying problem structure (Rush
 6944 et al., 2010). Das et al. (2014) apply this technique to FrameNet semantic role labeling,
 6945 obtaining an order-of-magnitude speedup over cplex.

6946 13.2.3 Neural semantic role labeling

6947 Neural network approaches to SRL have tended to treat it as a sequence labeling task,
 6948 using a labeling scheme such as the BIO notation, which we previously saw in named
 6949 entity recognition (§ 8.3). In this notation, the first token in a span of type Arg1 is labeled

¹¹<https://www.gnu.org/software/glpk/>

¹²<https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>

¹³<http://www.gurobi.com/>

6950 B-Arg1; all remaining tokens in the span are inside, and are therefore labeled I-Arg1.
 6951 Tokens outside any argument are labeled O. For example:

- 6952 (13.21) Asha taught Boyang's mom about algebra
 B-Arg0 Pred B-Arg2 I-Arg2 I-Arg2 B-Arg1 I-Arg1

Recurrent neural networks are a natural approach to this tagging task. For example, Zhou and Xu (2015) apply a deep bidirectional multilayer LSTM (see § 7.6) to PropBank semantic role labeling. In this model, each bidirectional LSTM serves as input for another, higher-level bidirectional LSTM, allowing complex non-linear transformations of the original input embeddings, $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_M]$. The hidden state of the final LSTM is $\mathbf{Z}^{(K)} = [\mathbf{z}_1^{(K)}, \mathbf{z}_2^{(K)}, \dots, \mathbf{z}_M^{(K)}]$. The “emission” score for each tag $Y_m = y$ is equal to the inner product $\theta_y \cdot \mathbf{z}_m^{(K)}$, and there is also a transition score for each pair of adjacent tags. The complete model can be written,

$$\mathbf{Z}^{(1)} = \text{BiLSTM}(\mathbf{X}) \quad [13.13]$$

$$\mathbf{Z}^{(i)} = \text{BiLSTM}(\mathbf{Z}^{(i-1)}) \quad [13.14]$$

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\text{argmax}} \sum_{m=1}^M \Theta^{(y)} \mathbf{z}_m^{(K)} + \psi_{y_{m-1}, y_m}. \quad [13.15]$$

6953 Note that the final step maximizes over the entire labeling \mathbf{y} , and includes a score for
 6954 each tag transition ψ_{y_{m-1}, y_m} . This combination of LSTM and pairwise potentials on tags
 6955 is an example of an LSTM-CRF. The maximization over \mathbf{y} is performed by the Viterbi
 6956 algorithm.

6957 This model strongly outperformed alternative approaches at the time, including constrained
 6958 decoding and convolutional neural networks.¹⁴ More recent work has combined recurrent
 6959 neural network models with constrained decoding, using the A^* search algorithm to search
 6960 over labelings that are feasible with respect to the constraints (He et al., 2017). This yields
 6961 small improvements over the method of Zhou and Xu (2015). He et al. (2017) obtain
 6962 larger improvements by creating an ensemble of SRL systems, each trained on an 80%
 6963 subsample of the corpus. The average prediction across this ensemble is more robust than
 6964 any individual model.

6965 13.3 Abstract Meaning Representation

6966 Semantic role labeling transforms the task of semantic parsing to a labeling task. Consider
 6967 the sentence,

¹⁴The successful application of convolutional neural networks to semantic role labeling by Collobert and Weston (2008) was an influential early result in the most recent wave of neural networks in natural language processing.

```
(w / want-01
:ARG0 (b / boy)
:ARG1 (g / go-02
:ARG0 b))
```

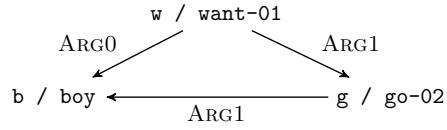


Figure 13.3: Two views of the AMR representation for the sentence The boy wants to go.

6968 (13.22) The boy wants to go.

6969 The PropBank semantic role labeling analysis is:

- 6970 • (Predicate : wants, Arg0 : the boy, Arg1 : to go)
- 6971 • (Predicate : go, Arg1 : the boy)

6972 The Abstract Meaning Representation (AMR) unifies this analysis into a graph structure,
6973 in which each node is a variable, and each edge indicates a concept (Banarescu et al., 2013).
6974 This can be written in two ways, as shown in Figure 13.3. On the left is the PENMAN
6975 notation (Matthiessen and Bateman, 1991), in which each set of parentheses introduces
6976 a variable. Each variable is an instance of a concept, which is indicated with the slash
6977 notation: for example, *w / want-01* indicates that the variable *w* is an instance of the
6978 concept *want-01*, which in turn refers to the PropBank frame for the first sense of the verb
6979 *want*. Relations are introduced with colons: for example, *:ARG0 (b / boy)* indicates a
6980 relation of type *arg0* with the newly-introduced variable *b*. Variables can be reused, so
6981 that when the variable *b* appears again as an argument to *g*, it is understood to refer to the
6982 same boy in both cases. This arrangement is indicated compactly in the graph structure
6983 on the right, with edges indicating concepts.

6984 One way in which AMR differs from PropBank-style semantic role labeling is that it
6985 reifies each entity as a variable: for example, the boy in (13.22) is reified in the variable *b*,
6986 which is reused as *Arg0* in its relationship with *w / want-01*, and as *Arg1* in its relationship
6987 with *g / go-02*. Reifying entities as variables also makes it possible to represent the
6988 substructure of noun phrases more explicitly. For example, Asha borrowed the algebra
6989 book would be represented as:

6990 (b / borrow-01
6991 :ARG0 (p / person
6992 :name (n / name
6993 :op1 "Asha"))
6994 :ARG1 (b2 / book
6995 :topic (a / algebra)))

6996 This indicates that the variable p is a person, whose name is the variable n; that name has
 6997 one token, the string Asha. Similarly, the variable b2 is a book, and the topic of b2 is a
 6998 variable a whose type is algebra. The relations name and topic are examples of non-core
 6999 roles, which are similar to adjunct modifiers in PropBank. However, AMR’s inventory is
 7000 more extensive, including more than 70 non-core roles, such as negation, time, manner,
 7001 frequency, and location. Lists and sequences — such as the list of tokens in a name — are
 7002 described using the roles op1, op2, etc.

7003 Another feature of AMR is that a semantic predicate can be introduced by any syntactic
 7004 element, as in the following examples from Banarescu et al. (2013):

- 7005 (13.23) The boy destroyed the room.
- 7006 (13.24) the destruction of the room by the boy ...
- 7007 (13.25) the boy’s destruction of the room ...

7008 All these examples have the same semantics in AMR,

7009 (d / destroy-01
 7010 :ARG0 (b / boy)
 7011 :ARG1 (r / room))

7012 The noun destruction is linked to the verb destroy, which is captured by the PropBank
 7013 frame destroy-01. This can happen with adjectives as well: in the phrase the attractive
 7014 spy, the adjective attractive is linked to the PropBank frame attract-01:

7015 (s / spy
 7016 :ARG0-of (a / attract-01))

7017 In this example, ARG0-of is an inverse relation, indicating that s is the ARG0 of the
 7018 predicate a. Inverse relations make it possible for all AMR parses to have a single root
 7019 concept, which should be the focus of the utterance.

7020 While AMR goes farther than semantic role labeling, it does not link semantically-related
 7021 frames such as buy/sell (as FrameNet does), does not handle quantification (as first-order
 7022 predicate calculus does), and makes no attempt to handle noun number and verb tense
 7023 (as PropBank does). A recent survey by Abend and Rappoport (2017) situates AMR with
 7024 respect to several other semantic representation schemes. Other linguistic features of AMR
 7025 are summarized in the original paper (Banarescu et al., 2013) and the tutorial slides by
 7026 Schneider et al. (2015).

7027 13.3.1 AMR Parsing

7028 Abstract Meaning Representation is not a labeling of the original text — unlike PropBank
7029 semantic role labeling, and most of the other tagging and parsing tasks that we have
7030 encountered thus far. The AMR for a given sentence may include multiple concepts for
7031 single words in the sentence: as we have seen, the sentence Asha likes algebra contains
7032 both person and name concepts for the word Asha. Conversely, words in the sentence
7033 may not appear in the AMR: in Boyang made a tour of campus, the light verb make
7034 would not appear in the AMR, which would instead be rooted on the predicate tour. As a
7035 result, AMR is difficult to parse, and even evaluating AMR parsing involves considerable
7036 algorithmic complexity (Cai and Yates, 2013).

7037 A further complexity is that AMR labeled datasets do not explicitly show the alignment
7038 between the AMR annotation and the words in the sentence. For example, the link between
7039 the word wants and the concept want-01 is not annotated. To acquire training data for
7040 learning-based parsers, it is therefore necessary to first perform an alignment between the
7041 training sentences and their AMR parses. Flanigan et al. (2014) introduce a rule-based
7042 parser, which links text to concepts through a series of increasingly high-recall steps.

7043 Graph-based parsing One family of approaches to AMR parsing is similar to the graph-based
7044 methods that we encountered in syntactic dependency parsing (chapter 11). For these
7045 systems (Flanigan et al., 2014), parsing is a two-step process:

- 7046 1. Concept identification (Figure 13.4a). This involves constructing concept subgraphs
7047 for individual words or spans of adjacent words. For example, in the sentence, Asha
7048 likes algebra, we would hope to identify the minimal subtree including just the concept
7049 like-01 for the word like, and the subtree ($p / \text{person} : \text{name} (n / \text{name} : \text{op1} \text{ Asha})$)
7050 for the word Asha.
- 7051 2. Relation identification (Figure 13.4b). This involves building a directed graph over
7052 the concepts, where the edges are labeled by the relation type. AMR imposes a
7053 number of constraints on the graph: all concepts must be included, the graph must
7054 be connected (there must be a path between every pair of nodes in the undirected
7055 version of the graph), and every node must have at most one outgoing edge of each
7056 type.

7057 Both of these problems are solved by structure prediction. Concept identification
7058 requires simultaneously segmenting the text into spans, and labeling each span with a
7059 graph fragment containing one or more concepts. This is done by computing a set of
7060 features for each candidate span s and concept labeling c , and then returning the labeling
7061 with the highest overall score.

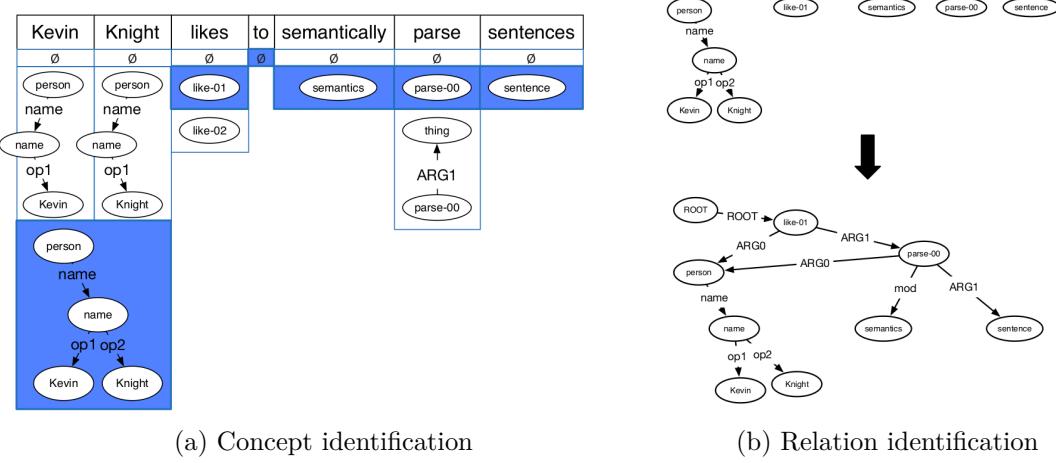


Figure 13.4: Subtasks for Abstract Meaning Representation parsing, from Schneider et al. (2015). [todo: permission]

7062 Relation identification can be formulated as search for the maximum spanning subgraph,
 7063 under a set of constraints. Each labeled edge has a score, which is computed from features
 7064 of the concepts. We then search for the set of labeled edges that maximizes the sum of
 7065 these scores, under the constraint that the resulting graph is a well-formed AMR (Flanigan
 7066 et al., 2014). This constrained search can be performed by optimization techniques such
 7067 as integer linear programming, as described in § 13.2.2.

7068 Transition-based parsing In many cases, AMR parses are structurally similar to syntactic
 7069 dependency parses. Figure 13.5 shows one such example. This motivates an alternative
 7070 approach to AMR parsing: modify the syntactic dependency parse until it looks like a good
 7071 AMR parse. Wang et al. (2015) propose a transition-based method, based on incremental
 7072 modifications to the syntactic dependency tree (transition-based dependency parsing is
 7073 discussed in § 11.3). At each step, the parser performs an action: for example, adding an
 7074 AMR relation label to the current dependency edge, swapping the direction of a syntactic
 7075 dependency edge, or cutting an edge and reattaching the orphaned subtree to a new parent.
 7076 The overall system is trained as a classifier, learning to choose the action as would be given
 7077 by an oracle that is capable of reproducing the ground-truth parse.

7078 13.4 Applications of Predicate-Argument Semantics

7079 Question answering Factoid questions have answers that are single words or phrases, such
 7080 as who discovered prions?, where was Barack Obama born?, and in what year did the Knicks
 7081 last win the championship? Semantic role labeling can be used to answer such questions,

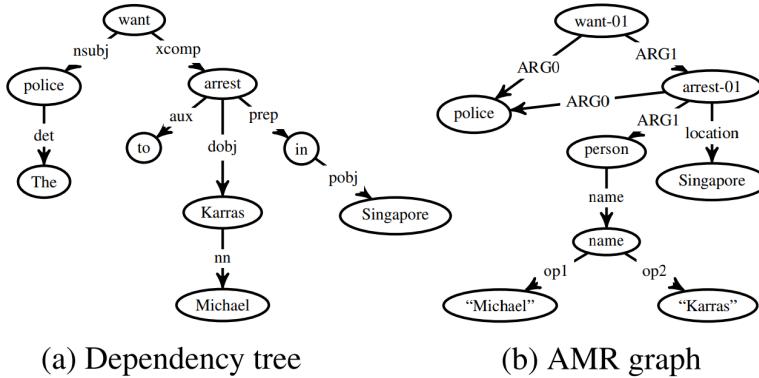


Figure 13.5: Syntactic dependency parse and AMR graph for the sentence The police want to arrest Michael Karras in Singapore (borrowed from Wang et al. (2015)) [todo: permission]

7082 by linking questions to sentences in a corpus of text. Shen and Lapata (2007) perform
 7083 FrameNet semantic role labeling on the query, and then construct a weighted bipartite
 7084 graph¹⁵ between FrameNet semantic roles and the words and phrases in the sentence. This
 7085 is done by first scoring all pairs of semantic roles and assignments, as shown in the top half
 7086 of Figure 13.6. They then find the bipartite edge cover, which is the minimum weighted
 7087 subset of edges such that each vertex has at least one edge, as shown in the bottom half
 7088 of Figure 13.6. After analyzing the question in this manner, Shen and Lapata then find
 7089 semantically-compatible sentences in the corpus, by performing graph matching on the
 7090 bipartite graphs for the question and candidate answer sentences. Finally, the expected
 7091 answer phrase in the question — typically the wh-word — is linked to a phrase in the
 7092 candidate answer source, and that phrase is returned as the answer.

7093 Relation extraction The task of relation extraction involves identifying pairs of entities
 7094 for which a given semantic relation holds (see § 17.2. For example, we might like to find all
 7095 pairs (i, j) such that i is the inventor-of j . PropBank semantic role labeling can be applied
 7096 to this task by identifying sentences whose verb signals the desired relation, and then
 7097 extracting Arg1 and Arg2 as arguments. (To fully solve this task, these arguments must
 7098 then be linked to entities, as described in chapter 17.) Christensen et al. (2010) compare a
 7099 semantic role labeling system against a simpler approach based on surface patterns (Banko
 7100 et al., 2007). They find that the SRL system is considerably more accurate, but that it
 7101 is several orders of magnitude slower. Conversely, Barnickel et al. (2009) apply SENNA,
 7102 a convolutional neural network SRL system (Collobert and Weston, 2008) to the task

¹⁵A bipartite graph is one in which the vertices can be divided into two disjoint sets, and every edge connects a vertex in one set to a vertex in the other.

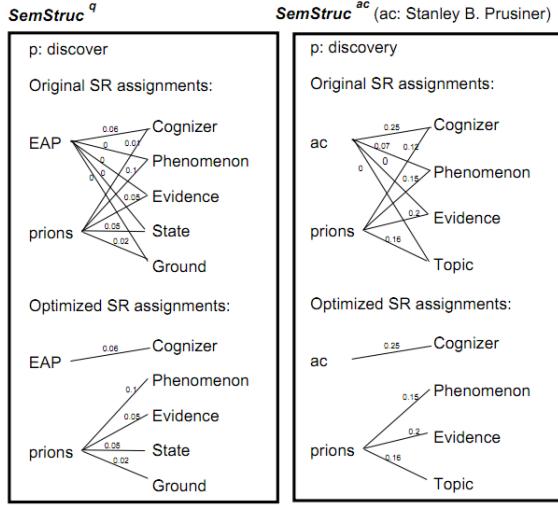


Figure 13.6: FrameNet semantic role labeling is used in factoid question answering, by aligning the semantic roles in the question (q) against those of sentences containing answer candidates (ac). “EAP” is the expected answer phrase, replacing the word who in the question. Figure reprinted from Shen and Lapata (2007) [todo: permission]

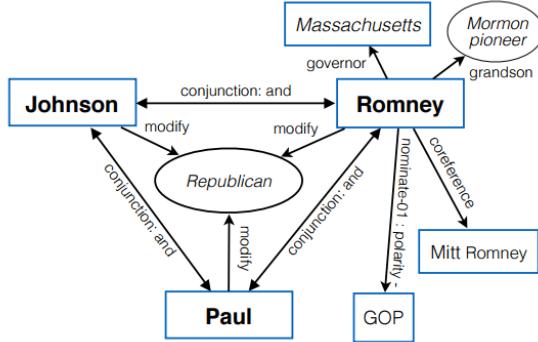


Figure 13.7: Fragment of AMR knowledge network for entity linking. Figure reprinted from Pan et al. (2015) [todo: permission]

7103 of identifying biomedical relations (e.g., which genes inhibit or activate each other). In
 7104 comparison with a strong baseline that applies a set of rules to syntactic dependency
 7105 structures (Fundel et al., 2007), the SRL system is faster but less accurate. One possible
 7106 explanation for these divergent results is that Fundel et al. compare against a baseline
 7107 which is carefully tuned for performance in a relatively narrow domain, while the system
 7108 of Banko et al. is designed to analyze text across the entire web.

Entity linking Another core task in information extraction is to link mentions of entities (e.g., Republican candidates like Romney, Paul, and Johnson ...) to entities in a knowledge base (e.g., Lyndon Johnson or Gary Johnson). This task, which is described in § 17.1, is often performed by examining nearby “collaborator” mentions — in this case, Romney and Paul. By jointly linking all 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).

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.
2. Find the PropBank framesets for share and hate at <http://verbs.colorado.edu/propbank/framesets-english-aliases/>, and rewrite your answers from the previous question, using the thematic roles Arg0, Arg1, and Arg2.
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 sure about the parse, you can try an online parser such as <http://nlp.stanford.edu:8080/parser/>.
4. Compute the dependency path features for Abigail and Max in each of the example sentences (13.26) and (13.28) in Question 1, with respect to the verb share. Again, if 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).
5. PropBank semantic role labeling includes reference arguments, such as,

(13.29) [Am-Loc The bed] on [R-Am-Loc which] I slept broke.¹⁶

The label R-Am-Loc indicates that word which is a reference to The bed, which expresses the location of the event. Reference arguments must have referents: the

¹⁶Example from 2013 NAACL tutorial slides by Shumin Wu

7140 tag R-Am-Loc can appear only when Am-Loc also appears in the sentence. Show
 7141 how to express this as a linear constraint, specifically for the tag R-Am-Loc. Be sure
 7142 to correctly handle the case in which neither Am-Loc nor R-Am-Loc appear in the
 7143 sentence.

- 7144 6. Explain how to express the constraints on semantic role labeling in Equation 13.8
 7145 and Equation 13.9 in the general form $\mathbf{Az} \geq \mathbf{b}$.
- 7146 7. Download the FrameNet sample data (<https://framenet.icsi.berkeley.edu/fndrupal/fulltextIndex>), and train a bag-of-words classifier to predict the frame that is evoked
 7147 by each verb in each example. Your classifier should build a bag-of-words from the
 7148 sentence in which the frame-evoking lexical unit appears. [todo: Somehow limit to
 7149 one or a few lexical units.] [todo: use NLTK if possible]
- 7150 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
 7151 LSTM to predict tags. You will have to convert the downloaded instances to a BIO
 7152 sequence labeling representation first.
- 7153 9. Produce the AMR annotations for the following examples:

- 7154 (13.30) The girl likes the boy.
- 7155 (13.31) The girl was liked by the boy.
- 7156 (13.32) Abigail likes Maxwell Aristotle.
- 7157 (13.33) The spy likes the attractive boy.
- 7158 (13.34) The girl doesn't like the boy.
- 7159 (13.35) The girl likes her dog.

7160 For (13.32), recall that multi-token names are created using op1, op2, etc. You will
 7161 need to consult Banarescu et al. (2013) for (13.34), and Schneider et al. (2015) for
 7162 (13.35). You may assume that her refers to the girl in this example.

- 7163 10. Using an off-the-shelf PropBank SRL system,¹⁷ build a simplified question answering
 7164 system in the style of Shen and Lapata (2007). Specifically, your system should do
 7165 the following:
 - 7166 • For each document in a collection, it should apply the semantic role labeler, and
 7167 should store the output as a tuple.

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/>).

- 7170 • For a question, your system should again apply the semantic role labeler. If
7171 any of the roles are filled by a wh-pronoun, you should mark that role as the
7172 expected answer phrase (EAP).
7173 • To answer the question, search for a stored tuple which matches the question as
7174 well as possible (same predicate, no incompatible semantic roles, and as many
7175 matching roles as possible). Align the EAP against its role filler in the stored
7176 tuple, and return this as the answer.

7177 To evaluate your system, download a set of three news articles on the same topic, and
7178 write down five factoid questions that should be answerable from the articles. See if
7179 your system can answer these questions correctly. (If this problem is assigned to an
7180 entire class, you can build a large-scale test set and compare various approaches.)

₇₁₈₁ Chapter 14

₇₁₈₂ Distributional and distributed semantics

₇₁₈₃ A recurring theme in natural language processing is the complexity of the mapping from
₇₁₈₄ words to meaning. In chapter 4, we saw that a single word form, like bank, can have
₇₁₈₅ multiple meanings; conversely, a single meaning may be created by multiple surface forms,
₇₁₈₆ a lexical semantic relationship known as synonymy. Despite this complex mapping between
₇₁₈₇ words and meaning, natural language processing systems usually rely on words as the
₇₁₈₈ basic unit of analysis. This is especially true in semantics: the logical and frame semantic
₇₁₈₉ methods from the previous two chapters rely on hand-crafted lexicons that map from
₇₁₉₀ words to semantic predicates. But how can we analyze texts that contain words that
₇₁₉₁ we haven't seen before? This chapter describes methods that learn representations of
₇₁₉₂ word meaning by analyzing unlabeled data, vastly improving the generalizability of natural
₇₁₉₃ language processing systems. The theory that makes it possible to acquire meaningful
₇₁₉₄ representations from unlabeled data is the distributional hypothesis.

₇₁₉₅ 14.1 The distributional hypothesis

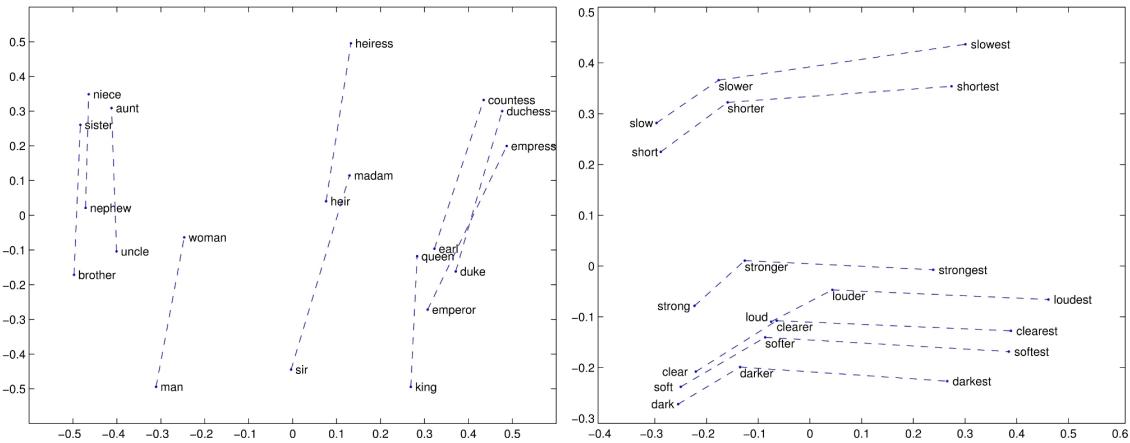
₇₁₉₆ Here's a word you may not know: tezgüino (the example is from Lin, 1998). If you do not
₇₁₉₇ know the meaning of tezgüino, then you are in the same situation as a natural language
₇₁₉₈ processing system when it encounters a word that did not appear in its training data. Now
₇₁₉₉ suppose you see that tezgüino is used in the following contexts:

- ₇₂₀₀ (14.1) A bottle of _____ is on the table.
₇₂₀₁ (14.2) Everybody likes _____.
₇₂₀₂ (14.3) Don't have _____ before you drive.
₇₂₀₃ (14.4) We make _____ out of corn.

₇₂₀₄ What other words fit into these contexts? How about: loud, motor oil, tortillas, choices,
₇₂₀₅ wine? Each row of Table 14.1 is a vector that summarizes the contextual properties for

	(14.1)	(14.2)	(14.3)	(14.4)	...
tezgüino	1	1	1	1	
loud	0	0	0	0	
motor oil	1	0	0	1	
tortillas	0	1	0	1	
choices	0	1	0	0	
wine	1	1	1	0	

Table 14.1: Distributional statistics for tezgüino and five related terms

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?])

7206 each word, with a value of one for contexts in which the word can appear, and a value of
 7207 zero for contexts in which it cannot. Based on these vectors, we can conclude: wine is very
 7208 similar to tezgüino; motor oil and tortillas are fairly similar to tezgüino; loud is completely
 7209 different.

7210 These vectors, which we will call word representations, describe the distributional
 7211 properties of each word. Does vector similarity imply semantic similarity? This is the
 7212 distributional hypothesis, stated by Firth (1957) as: “You shall know a word by the
 7213 company it keeps.” The distributional hypothesis has stood the test of time: distributional
 7214 statistics are a core part of language technology today, because they make it possible to
 7215 leverage large amounts of unlabeled data to learn about rare words that do not appear in
 7216 labeled training data.

7217 Distributional statistics have a striking ability to capture lexical semantic relationships
7218 such as analogies. Figure 14.1 shows two examples, based on two-dimensional projections
7219 of distributional word embeddings, discussed later in this chapter. In each case, word-pair
7220 relationships correspond to regular linear patterns in this two dimensional space. No
7221 labeled data about the nature of these relationships was required to identify this underlying
7222 structure.

7223 Distributional semantics are computed from context statistics. Distributed semantics
7224 are a related but distinct idea: that meaning can be represented by numerical vectors
7225 rather than symbolic structures. Distributed representations are often estimated from
7226 distributional statistics, as in latent semantic analysis and word2vec, described later in
7227 this chapter. However, distributed representations can also be learned in a supervised
7228 fashion from labeled data, as in the neural classification models encountered in chapter 3.

7229 14.2 Design decisions for word representations

7230 There are many approaches for computing word representations, but most can be distinguished
7231 on three main dimensions: the nature of the representation, the source of contextual
7232 information, and the estimation procedure.

7233 14.2.1 Representation

7234 Today, the dominant word representations are k -dimensional vectors of real numbers,
7235 known as word embeddings. (The name is due to the fact that each discrete word is
7236 embedded in a continuous vector space.) This representation dates back at least to the late
7237 1980s (Deerwester et al., 1990), and is used in popular techniques such as word2vec (Mikolov
7238 et al., 2013).

7239 Word embeddings are well suited for neural networks, where they can be plugged in as
7240 inputs. They can also be applied in linear classifiers and structure prediction models (Turian
7241 et al., 2010), although it can be difficult to learn linear models that employ real-valued
7242 features (Kummerfeld et al., 2015). A popular alternative is bit-string representations,
7243 such as Brown clusters (§ 14.4), in which each word is represented by a variable-length
7244 sequence of zeros and ones (Brown et al., 1992).

7245 Another representational question is whether to estimate one embedding per surface
7246 form (e.g., bank), or to estimate distinct embeddings for each word sense or synset.
7247 Intuitively, if word representations are to capture the meaning of individual words, then
7248 words with multiple meanings should have multiple embeddings. This can be achieved by
7249 integrating unsupervised clustering with word embedding estimation (Huang and Yates,
7250 2012; Li and Jurafsky, 2015). However, Arora et al. (2016) argue that it is unnecessary to
7251 model distinct word senses explicitly, because the embeddings for each surface form are a
7252 linear combination of the embeddings of the underlying senses.

The moment one learns English, complications set in (Alfau, 1999)	
Brown Clusters (Brown et al., 1992)	{one}
word2vec (Mikolov et al., 2013) ($h = 2$)	{moment, one, English, complications}
Structured word2vec (Ling et al., 2015) ($h = 2$)	{(moment, -2), (one, -1), (English, +1), (complications, +2)}
Dependency contexts (Levy and Goldberg, 2014)	{(one, 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.

7253 14.2.2 Context

7254 The distributional hypothesis says that word meaning is related to the “contexts” in which
 7255 the word appears, but context can be defined in many ways. In the tezgüino example,
 7256 contexts are entire sentences, but in practice there are far too many sentences. At the
 7257 opposite extreme, the context could be defined as the immediately preceding word; this is
 7258 the context considered in Brown clusters. word2vec takes an intermediate approach, using
 7259 local neighborhoods of words (e.g., $h = 5$) as contexts (Mikolov et al., 2013). Contexts
 7260 can also be much larger: for example, in latent semantic analysis, each word’s context
 7261 vector includes an entry per document, with a value of one if the word appears in the
 7262 document (Deerwester et al., 1990); in explicit semantic analysis, these documents are
 7263 Wikipedia pages (Gabrilovich and Markovitch, 2007).

7264 Words in context can be labeled by their position with respect to the target word w_m
 7265 (e.g., two words before, one word after), which makes the resulting word representations
 7266 more sensitive to syntactic differences (Ling et al., 2015). Another way to incorporate
 7267 syntax is to perform parsing as a preprocessing step, and then form context vectors from
 7268 the dependency edges (Levy and Goldberg, 2014) or predicate-argument relations (Lin,
 7269 1998). The resulting context vectors for several of these methods are shown in Table 14.2.

7270 The choice of context has a profound effect on the resulting representations, which
 7271 can be viewed in terms of word similarity. Applying latent semantic analysis (§ 14.3) to
 7272 contexts of size $h = 2$ and $h = 30$ yields the following nearest-neighbors for the word dog:¹

¹The example is from lecture slides by Marco Baroni, Alessandro Lenci, and Stefan Evert, who applied latent semantic analysis to the British National Corpus. You can find an online demo here: <http://clic.cimec.unitn.it/infomap-query/>

- 7273 • ($h = 2$): cat, horse, fox, pet, rabbit, pig, animal, mongrel, sheep, pigeon
 7274 • ($h = 30$): kennel, puppy, pet, bitch, terrier, rottweiler, canine, cat, to bark, Alsatian

7275 Which word list is better? Each word in the $h = 2$ list is an animal, reflecting the fact
 7276 that locally, the word dog tends to appear in the same contexts as other animal types (e.g.,
 7277 pet the dog, feed the dog). In the $h = 30$ list, nearly everything is dog-related, including
 7278 specific breeds such as rottweiler and Alsatian. The list also includes words that are not
 7279 animals (kennel), and in one case (to bark), is not a noun at all. The 2-word context
 7280 window is more sensitive to syntax, while the 30-word window is more sensitive to topic.

7281 14.2.3 Estimation

7282 Word embeddings are estimated by optimizing some objective: the likelihood of a set of
 7283 unlabeled data (or a closely related quantity), or the reconstruction of a matrix of context
 7284 counts, similar to Table 14.1.

7285 Maximum likelihood estimation Likelihood-based optimization is derived from the objective
 7286 $\log p(\mathbf{w}; \mathbf{U})$, where $\mathbf{U} \in \mathbb{R} K \times V$ is matrix of word embeddings, and $\mathbf{w} = \{w_m\}_{m=1}^M$ is a
 7287 corpus, represented as a list of M tokens. Recurrent neural network language models (§ 6.3)
 7288 optimize this objective directly, backpropagating to the input word embeddings through
 7289 the recurrent structure. However, state-of-the-art word embeddings employ huge corpora
 7290 with hundreds of billions of tokens, and recurrent architectures are difficult to scale to
 7291 such data. As a result, likelihood-based word embeddings are usually based on simplified
 7292 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,

$$\min_{\mathbf{u}, \mathbf{v}} \|\mathbf{C} - \tilde{\mathbf{C}}(\mathbf{u}, \mathbf{v})\|_F, \quad [14.1]$$

7293 where $\tilde{\mathbf{C}}(\mathbf{u}, \mathbf{v})$ is the approximate reconstruction resulting from the embeddings \mathbf{u} and \mathbf{v} ,
 7294 and $\|\mathbf{X}\|_F$ indicates the Frobenius norm, $\sum_{i,j} x_{i,j}^2$. Rather than factoring the matrix of
 7295 word-context counts directly, it is often helpful to transform these counts using information-theoretic
 7296 metrics such as pointwise mutual information (pmi), described in the next section.

7297 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),

$$\min_{\mathbf{U} \in \mathbb{R}^{V \times K}, \mathbf{S} \in \mathbb{R}^{K \times K}, \mathbf{V} \in \mathbb{R}^{|\mathcal{C}| \times K}} \|\mathbf{C} - \mathbf{U}\mathbf{S}\mathbf{V}^\top\|_F \quad [14.2]$$

$$\text{s.t. } \mathbf{U}^\top \mathbf{U} = \mathbb{I} \quad [14.3]$$

$$\mathbf{V}^\top \mathbf{V} = \mathbb{I} \quad [14.4]$$

$$\forall i \neq j, \mathbf{S}_{i,j} = 0, \quad [14.5]$$

7298 where V is the size of the vocabulary, $|\mathcal{C}|$ is the number of contexts, and K is size of the
 7299 resulting embeddings, which are set equal to the rows of the matrix \mathbf{U} . The matrix \mathbf{S} is
 7300 constrained to be diagonal (these diagonal elements are called the singular values), and
 7301 the columns of the product $\mathbf{S}\mathbf{V}^\top$ provide descriptions of the contexts. Each element $c_{i,j}$ is
 7302 then reconstructed as a bilinear product,

$$c_{i,j} \approx \sum_{k=1}^K u_{i,k} s_k v_{j,k}. \quad [14.6]$$

7303 The objective is to minimize the sum of squared approximation errors. The orthonormality
 7304 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
 7305 uncorrelated, so that each dimension conveys unique information. Efficient implementations
 7306 of truncated singular value decomposition are available in numerical computing packages
 7307 such as scipy and matlab.²

Latent semantic analysis is most effective when the count matrix is transformed before the application of SVD. One such transformation is pointwise mutual information (pmi; Church and Hanks, 1990), which captures the degree of association between word i and context j ,

$$\text{pmi}(i, j) = \log \frac{p(i, j)}{p(i)p(j)} = \log \frac{p(i | j)p(j)}{p(i)p(j)} = \log \frac{p(i | j)}{p(i)} \quad [14.7]$$

$$= \log \text{count}(i, j) - \log \sum_{i'=1}^V \text{count}(i', j) \quad [14.8]$$

$$- \log \sum_{j' \in \mathcal{C}} \text{count}(i, j') + \log \sum_{i'=1}^V \sum_{j' \in \mathcal{C}} \text{count}(i', j'). \quad [14.9]$$

²An important implementation detail is to represent \mathbf{C} as a sparse matrix, so that the storage cost is equal to the number of non-zero entries, rather than the size $V \times |\mathcal{C}|$.

7308 The pointwise mutual information can be viewed as the logarithm of the ratio of the
 7309 conditional probability of word i in context j to the marginal probability of word i in
 7310 all contexts. When word i is statistically associated with context j , the ratio will be
 7311 greater than one, so $\text{pmi}(i, j) > 0$. The pmi transformation focuses latent semantic analysis
 7312 on reconstructing strong word-context associations, rather than on reconstructing large
 7313 counts.

7314 The pmi is negative when a word and context occur together less often than if they were
 7315 independent, but such negative correlations are unreliable because counts of rare events
 7316 have high variance. Furthermore, the pmi is undefined when $\text{count}(i, j) = 0$. One solution
 7317 to these problems is to use the Positive pmi (ppmi),

$$\text{ppmi}(i, j) = \begin{cases} \text{pmi}(i, j), & p(i | j) > p(i) \\ 0, & \text{otherwise.} \end{cases} \quad [14.10]$$

7318 Bullinaria and Levy (2007) compare a range of matrix transformations for latent semantic
 7319 analysis, using a battery of tasks related to word meaning and word similarity (for more on
 7320 evaluation, see § 14.6). They find that PPMI-based latent semantic analysis yields strong
 7321 performance on a battery of tasks related to word meaning: for example, PPMI-based
 7322 LSA vectors can be used to solve multiple-choice word similarity questions from the Test
 7323 of English as a Foreign Language (TOEFL), obtaining 85% accuracy.

7324 14.4 Brown clusters

7325 Learning algorithms like perceptron and conditional random fields often perform better
 7326 with discrete feature vectors. A simple way to obtain discrete representations from distributional
 7327 statistics is by clustering (§ 5.1.1), so that words in the same cluster have similar distributional
 7328 statistics. This can help in downstream tasks, by sharing features between all words in the
 7329 same cluster. However, there is an obvious tradeoff: if the number of clusters is too small,
 7330 the words in each cluster will not have much in common; if the number of clusters is too
 7331 large, then the learner will not see enough examples from each cluster to generalize.

7332 A solution to this problem is hierarchical clustering: using the distributional statistics
 7333 to induce a tree-structured representation. Fragments of Brown cluster trees are shown
 7334 in Figure 14.2 and Table 14.3. Each word's representation consists of a binary string
 7335 describing a path through the tree: 0 for taking the left branch, and 1 for taking the right
 7336 branch. In the subtree in the upper right of the figure, the representation of the word
 7337 conversation is 10; the representation of the word assessment is 0001. Bitstring prefixes
 7338 capture similarity at varying levels of specificity, and it is common to use the first eight,
 7339 twelve, sixteen, and twenty bits as features in tasks such as named entity recognition (Miller
 7340 et al., 2004) and dependency parsing (Koo et al., 2008).

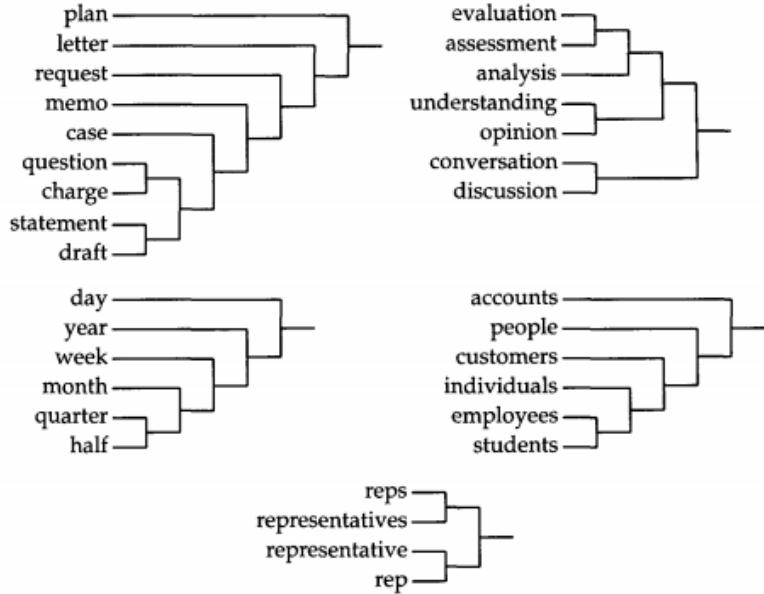


Figure 14.2: Some subtrees produced by bottom-up Brown clustering (Miller et al., 2004) on news text [todo: permission]

Hierarchical trees can be induced from a likelihood-based objective, using a discrete latent variable $k_i \in \{1, 2, \dots, K\}$ to represent the cluster of word i :

$$\log p(\mathbf{w}; \mathbf{k}) \approx \sum_{m=1}^M \log p(w_m | w_{m-1}; \mathbf{k}) \quad [14.11]$$

$$\triangleq \sum_{m=1}^M \log p(w_m | k_{w_m}) + \log p(k_{w_m} | k_{w_{m-1}}). \quad [14.12]$$

⁷³⁴¹ This is similar to a hidden Markov model, with the crucial difference that each word can
⁷³⁴² be emitted from only a single cluster: $\forall k \neq k_{w_m}, p(w_m | k) = 0$.

Using the objective in Equation 14.12, the Brown clustering tree can be constructed from the bottom up: begin with each word in its own cluster, and incrementally merge clusters until only a single cluster remains. At each step, we merge the pair of clusters such that the objective in Equation 14.12 is maximized. Although the objective seems to involve a sum over the entire corpus, the score for each merger can be computed from the cluster-to-cluster co-occurrence counts. These counts can be updated incrementally as the clustering proceeds. The optimal merge at each step can be shown to maximize the average

bitstring	ten most frequent words
011110100111	excited thankful grateful stoked pumped anxious hyped psyched exited geeked
01111010100	talking talkin complaining talkn bitching tlkn tlkin bragging raving +k
011110101010	thinking thinkin dreaming worrying thinkn speakin reminiscing dreamin daydreaming fantasizing
011110101011	saying sayin suggesting stating sayn jokin talmbout implying insisting 5'2
011110101100	wonder dunno wondered duno donno dno do no wonda wounder dunnoe
011110101101	wondering wonders debating deciding pondering unsure wonderin debatin woundering wondern
011110101110	sure suree suuure suure sure- surre sures shuree

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.

mutual information,

$$I(\mathbf{k}) = \sum_{k_1=1}^K \sum_{k_2=1}^K p(k_1, k_2) \times \text{pmi}(k_1, k_2) \quad [14.13]$$

$$p(k_1, k_2) = \frac{\text{count}(k_1, k_2)}{\sum_{k_{1'}=1}^K \sum_{k_{2'}=1}^K \text{count}(k_{1'}, k_{2'})},$$

where $p(k_1, k_2)$ is the joint probability of a bigram involving a word in cluster k_1 followed by a word in k_2 . This probability and the pmi are both computed from the co-occurrence counts between clusters. After each merger, the co-occurrence vectors for the merged clusters are simply added up, so that the next optimal merger can be found efficiently.

This bottom-up procedure requires iterating over the entire vocabulary, and evaluating K_t^2 possible mergers at each step, where K_t is the current number of clusters at step t of the algorithm. Furthermore, computing the score for each merger involves a sum over K_t^2 clusters. The maximum number of clusters is $K_0 = V$, which occurs when every word is in its own cluster at the beginning of the algorithm. The time complexity is thus $\mathcal{O}(V^5)$.

Algorithm 17 Exchange clustering algorithm. Assumes that words are sorted by frequency, and that MaxMI finds the cluster pair whose merger maximizes the mutual information, as defined in Equation 14.13.

```

procedure ExchangeClustering( $\{\text{count}(\cdot, \cdot)\}, K$ )
    for  $i \in 1 \dots K$  do                                 $\triangleright$  Initialization
         $k_i \leftarrow i, \quad i = 1, 2, \dots, K$ 
        for  $j \in 1 \dots K$  do
             $c_{i,j} \leftarrow \text{count}(i, j)$ 
         $\tau \leftarrow \{(i)\}_{i=1}^K$ 
        for  $i \in \{K+1, K+2, \dots, V\}$  do            $\triangleright$  Iteratively add each word to the clustering
             $\tau \leftarrow \tau \cup (i)$ 
            for  $k \in \tau$  do
                 $c_{k,i} \leftarrow \text{count}(k, i)$ 
                 $c_{i,k} \leftarrow \text{count}(i, k)$ 
             $\hat{i}, \hat{j} \leftarrow \text{MaxMI}(\mathbf{C})$ 
             $\tau, \mathbf{C} \leftarrow \text{Merge}(\hat{i}, \hat{j}, \mathbf{C}, \tau)$ 
        repeat                                          $\triangleright$  Merge the remaining clusters into a tree
             $\hat{i}, \hat{j} \leftarrow \text{MaxMI}(\mathbf{C}, \tau)$ 
             $\tau, \mathbf{C} \leftarrow \text{Merge}(\hat{i}, \hat{j}, \mathbf{C}, \tau)$ 
        until  $|\tau| = 1$ 
        return  $\tau$ 
procedure Merge( $i, j, \mathbf{C}, \tau$ )
     $\tau \leftarrow \tau \setminus i \setminus j \cup (i, j)$            $\triangleright$  Merge the clusters in the tree
    for  $k \in \tau$  do                                      $\triangleright$  Aggregate the counts across the merged clusters
         $c_{k,(i,j)} \leftarrow c_{k,i} + c_{k,j}$ 
         $c_{(i,j),k} \leftarrow c_{i,k} + c_{j,k}$ 
    return  $\tau, \mathbf{C}$ 

```

7352 To avoid this complexity, practical implementations use a heuristic approximation called
 7353 exchange clustering. The K most common words are placed in clusters of their own at the
 7354 beginning of the process. We then consider the next most common word, and merge it
 7355 with one of the existing clusters. This continues until the entire vocabulary has been
 7356 incorporated, at which point the K clusters are merged down to a single cluster, forming
 7357 a tree. The algorithm never considers more than $K+1$ clusters at any step, and the
 7358 complexity is $\mathcal{O}(VK + V \log V)$, with the second term representing the cost of sorting the
 7359 words at the beginning of the algorithm.

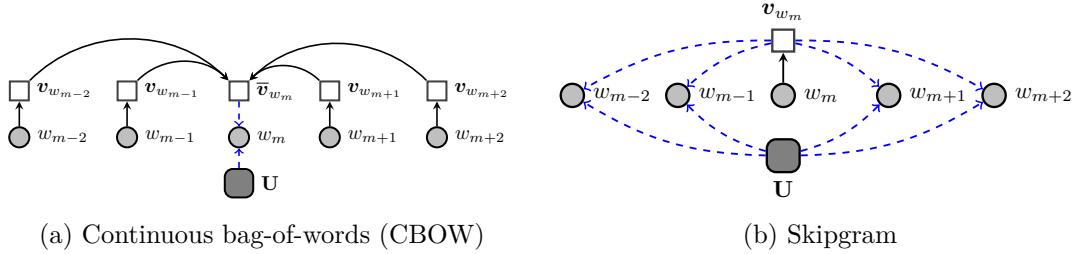


Figure 14.3: The CBOW and skipgram variants of word2vec. The parameter \mathbf{U} is the matrix of word embeddings, and each \mathbf{v}_m is the context embedding for word w_m .

7360 14.5 Neural word embeddings

7361 Neural word embeddings combine aspects of the previous two methods: like latent semantic
 7362 analysis, they are a continuous vector representation; like Brown clusters, they are trained
 7363 from a likelihood-based objective. Let the vector \mathbf{u}_i represent the K -dimensional embedding
 7364 for word i , and let \mathbf{v}_j represent the K -dimensional embedding for context j . The inner
 7365 product $\mathbf{u}_i \cdot \mathbf{v}_j$ represents the compatibility between word i and context j . By incorporating
 7366 this inner product into an approximation to the log-likelihood of a corpus, it is possible to
 7367 estimate both parameters by backpropagation. word2vec (Mikolov et al., 2013) includes
 7368 two such approximations: continuous bag-of-words (CBOW) and skipgrams.

7369 14.5.1 Continuous bag-of-words (CBOW)

7370 In recurrent neural network language models, each word w_m is conditioned on a recurrently-updated
 7371 state vector, which is based on word representations going all the way back to the beginning
 7372 of the text. The continuous bag-of-words (CBOW) model is a simplification: the local
 7373 context is computed as an average of embeddings for words in the immediate neighborhood
 7374 $m - h, m - h + 1, \dots, m + h - 1, m + h$,

$$\bar{\mathbf{v}}_m = \frac{1}{2h} \sum_{n=1}^h \mathbf{v}_{w_{m+n}} + \mathbf{v}_{w_{m-n}}. \quad [14.14]$$

7375 Thus, CBOW is a bag-of-words model, because the order of the context words does not
 7376 matter; it is continuous, because rather than conditioning on the words themselves, we
 7377 condition on a continuous vector constructed from the word embeddings. The parameter
 7378 h determines the neighborhood size, which Mikolov et al. (2013) set to $h = 4$.

The CBOW model optimizes an approximation to the corpus log-likelihood,

$$\log p(\mathbf{w}) \approx \sum_{m=1}^M \log p(w_m | w_{m-h}, w_{m-h+1}, \dots, w_{m+h-1}, w_{m+h}) \quad [14.15]$$

$$= \sum_{m=1}^M \log \frac{\exp(\mathbf{u}_{w_m} \cdot \bar{\mathbf{v}}_m)}{\sum_{j=1}^V \exp(\mathbf{u}_j \cdot \bar{\mathbf{v}}_m)} \quad [14.16]$$

$$= \sum_{m=1}^M \mathbf{u}_{w_m} \cdot \bar{\mathbf{v}}_m - \log \sum_{j=1}^V \exp(\mathbf{u}_j \cdot \bar{\mathbf{v}}_m). \quad [14.17]$$

7379 14.5.2 Skipgrams

In the CBOW model, words are predicted from their context. In the skipgram model, the context is predicted from the word, yielding the objective:

$$\log p(\mathbf{w}) \approx \sum_{m=1}^M \sum_{n=1}^{h_m} \log p(w_{m-n} | w_m) + \log p(w_{m+n} | w_m) \quad [14.18]$$

$$= \sum_{m=1}^M \sum_{n=1}^{h_m} \log \frac{\exp(\mathbf{u}_{w_{m-n}} \cdot \mathbf{v}_{w_m})}{\sum_{j=1}^V \exp(\mathbf{u}_j \cdot \mathbf{v}_{w_m})} + \log \frac{\exp(\mathbf{u}_{w_{m+n}} \cdot \mathbf{v}_{w_m})}{\sum_{j=1}^V \exp(\mathbf{u}_j \cdot \mathbf{v}_{w_m})} \quad [14.19]$$

$$= \sum_{m=1}^M \sum_{n=1}^{h_m} \mathbf{u}_{w_{m-n}} \cdot \mathbf{v}_{w_m} + \mathbf{u}_{w_{m+n}} \cdot \mathbf{v}_{w_m} - 2 \log \sum_{j=1}^V \exp(\mathbf{u}_j \cdot \mathbf{v}_{w_m}). \quad [14.20]$$

7380 In the skipgram approximation, each word is generated multiple times; each time it is
 7381 conditioned only on a single word. This makes it possible to avoid averaging the word
 7382 vectors, as in the CBOW model. The local neighborhood size h_m is randomly sampled from
 7383 a uniform categorical distribution over the range $\{1, 2, \dots, h_{\max}\}$; Mikolov et al. (2013) set
 7384 $h_{\max} = 10$. Because the neighborhood grows outward with h , this approach has the effect
 7385 of weighting near neighbors more than distant ones. Skipgram performs better on most
 7386 evaluations than CBOW (see § 14.6 for details of how to evaluate word representations),
 7387 but CBOW is faster to train (Mikolov et al., 2013).

7388 14.5.3 Computational complexity

7389 The word2vec models can be viewed as an efficient alternative to recurrent neural network
 7390 language models, which involve a recurrent state update whose time complexity is quadratic
 7391 in the size of the recurrent state vector. CBOW and skipgram avoid this computation, and
 7392 incur only a linear time complexity in the size of the word and context representations.
 7393 However, all three models compute a normalized probability over word tokens; a naïve
 7394 implementation of this probability requires summing over the entire vocabulary. The time

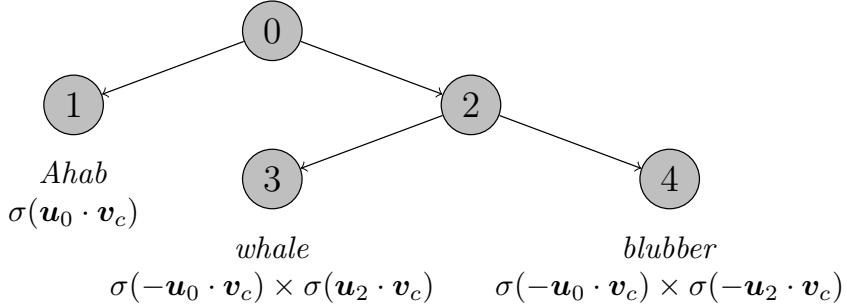


Figure 14.4: A fragment of a hierarchical softmax tree. The probability of each word is computed as a product of probabilities of local branching decisions in the tree.

complexity of this sum is $\mathcal{O}(V \times K)$, which dominates all other computational costs. There are two solutions: hierarchical softmax, a tree-based computation that reduces the cost to a logarithm of the size of the vocabulary; and negative sampling, an approximation that eliminates the dependence on vocabulary size. Both methods are also applicable to RNN language models.

14.5.3.1 Hierarchical softmax

In Brown clustering, the vocabulary is organized into a binary tree. Mnih and Hinton (2008) show that the normalized probability over words in the vocabulary can be reparametrized as a probability over paths through such a tree. This hierarchical softmax probability is computed as a product of binary decisions over whether to move left or right through the tree, with each binary decision represented as a sigmoid function of the inner product between the context embedding \mathbf{v}_c and an output embedding associated with the node \mathbf{u}_n ,

$$\Pr(\text{left at } n \mid c) = \sigma(\mathbf{u}_n \cdot \mathbf{v}_c) \quad [14.21]$$

$$\Pr(\text{right at } n \mid c) = 1 - \sigma(\mathbf{u}_n \cdot \mathbf{v}_c) = \sigma(-\mathbf{u}_n \cdot \mathbf{v}_c), \quad [14.22]$$

where σ refers to the sigmoid function, $\sigma(x) = \frac{1}{1+\exp(-x)}$. The range of the sigmoid is the interval $(0, 1)$, and $1 - \sigma(x) = \sigma(-x)$.

As shown in Figure 14.4, the probability of generating each word is redefined as the product of the probabilities across its path. The sum of all such path probabilities is guaranteed to be one, for any context vector $\mathbf{v}_c \in \mathbb{R}^K$. In a balanced binary tree, the depth is logarithmic in the number of leaf nodes, and thus the number of multiplications is equal to $\mathcal{O}(\log V)$. The number of non-leaf nodes is equal to $\mathcal{O}(2V - 1)$, so the number of parameters to be estimated increases by only a small multiple. The tree can be constructed using an incremental clustering procedure similar to hierarchical Brown clusters (Mnih and Hinton, 2008), or by using the Huffman (1952) encoding algorithm for lossless compression.

7411 14.5.3.2 Negative sampling

Likelihood-based methods are computationally intensive because each probability must be normalized over the vocabulary. These probabilities are based on scores for each word in each context, and it is possible to design an alternative objective that is based on these scores more directly: we seek word embeddings that maximize the score for the word that was really observed in each context, while minimizing the scores for a set of randomly selected negative samples:

$$\psi(i, j) = \log \sigma(\mathbf{u}_i \cdot \mathbf{v}_j) + \sum_{i' \in \mathcal{W}_{\text{neg}}} \log(1 - \sigma(\mathbf{u}_{i'} \cdot \mathbf{v}_j)), \quad [14.23]$$

7412 where $\psi(i, j)$ is the score for word i in context j , and \mathcal{W}_{neg} is the set of negative samples.
 7413 The objective is to maximize the sum over the corpus, $\sum_{m=1}^M \psi(w_m, c_m)$, where w_m is token
 7414 m and c_m is the associated context.

7415 The set of negative samples \mathcal{W}_{neg} is obtained by sampling from a unigram language
 7416 model. Mikolov et al. (2013) construct this unigram language model by exponentiating the
 7417 empirical word probabilities, setting $\hat{p}(i) \propto (\text{count}(i))^{\frac{3}{4}}$. This has the effect of redistributing
 7418 probability mass from common to rare words. The number of negative samples increases
 7419 the time complexity of training by a constant factor. Mikolov et al. (2013) report that
 7420 5-20 negative samples works for small training sets, and that two to five samples suffice for
 7421 larger corpora.

7422 14.5.4 Word embeddings as matrix factorization

7423 The negative sampling objective in Equation 14.23 can be justified as an efficient approximation
 7424 to the log-likelihood, but it is also closely linked to the matrix factorization objective
 7425 employed in latent semantic analysis. For a matrix of word-context pairs in which all
 7426 counts are non-zero, negative sampling is equivalent to factorization of the matrix \mathbf{M} ,
 7427 where $M_{ij} = \text{PMI}(i, j) - \log k$: each cell in the matrix is equal to the pointwise mutual
 7428 information of the word and context, shifted by $\log k$, with k equal to the number of negative
 7429 samples (Levy and Goldberg, 2014). For word-context pairs that are not observed in the
 7430 data, the pointwise mutual information is $-\infty$, but this can be addressed by considering
 7431 only PMI values that are greater than $\log k$, resulting in a matrix of shifted positive
 7432 pointwise mutual information,

$$M_{ij} = \max(0, \text{PMI}(i, j) - \log k). \quad [14.24]$$

7433 Word embeddings are obtained by factoring this matrix with truncated singular value
 7434 decomposition.

GloVe (“global vectors”) are a closely related approach (Pennington et al., 2014), in which the matrix to be factored is constructed from log co-occurrence counts, $M_{ij} =$

word 1	word 2	similarity
love	sex	6.77
stock	jaguar	0.92
money	cash	9.15
development	issue	3.97
lad	brother	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)).

$\log \text{count}(i, j)$. The word embeddings are estimated by minimizing the sum of squares,

$$\begin{aligned} \min_{\mathbf{u}, \mathbf{v}, b, \tilde{b}} \quad & \sum_{j=1}^V \sum_{j \in \mathcal{C}} f(M_{ij}) (\widehat{\log M_{ij}} - \log M_{ij})^2 \\ \text{s.t.} \quad & \widehat{\log M_{ij}} = \mathbf{u}_i \cdot \mathbf{v}_j + b_i + \tilde{b}_j, \end{aligned} \quad [14.25]$$

where b_i and \tilde{b}_j are offsets for word i and context j , which are estimated jointly with the embeddings \mathbf{u} and \mathbf{v} . The weighting function $f(M_{ij})$ is set to be zero at $M_{ij} = 0$, thus avoiding the problem of taking the logarithm of zero counts; it saturates at $M_{ij} = m_{\max}$, thus avoiding the problem of overcounting common word-context pairs. This heuristic turns out to be critical to the method’s performance.

The time complexity of sparse matrix reconstruction is determined by the number of non-zero word-context counts. Pennington et al. (2014) show that this number grows sublinearly with the size of the dataset: roughly $\mathcal{O}(N^{0.8})$ for typical English corpora. In contrast, the time complexity of word2vec is linear in the corpus size. Computing the co-occurrence counts also requires linear time in the size of the corpus, but this operation can easily be parallelized using MapReduce-style algorithms (Dean and Ghemawat, 2008).

14.6 Evaluating word embeddings

Distributed word representations can be evaluated in two main ways. Intrinsic evaluations test whether the representations cohere with our intuitions about word meaning. Extrinsic evaluations test whether they are useful for downstream tasks, such as sequence labeling.

14.6.1 Intrinsic evaluations

A basic question for word embeddings is whether the similarity of words i and j is reflected in the similarity of the vectors \mathbf{u}_i and \mathbf{u}_j . Cosine similarity is typically used to compare

7453 two word embeddings,

$$\cos(\mathbf{u}_i, \mathbf{u}_j) = \frac{\mathbf{u}_i \cdot \mathbf{u}_j}{\|\mathbf{u}_i\|_2 \times \|\mathbf{u}_j\|_2}. \quad [14.26]$$

7454 For any embedding method, we can evaluate whether the cosine similarity of word embeddings
 7455 is correlated with human judgments of word similarity. The WS-353 dataset (Finkelstein
 7456 et al., 2002) includes similarity scores for 353 word pairs (Table 14.4). To test the accuracy
 7457 of embeddings for rare and morphologically complex words, Luong et al. (2013) introduce
 7458 a dataset of “rare words.” Outside of English, word similarity resources are limited, mainly
 7459 consisting of translations of WS-353.

7460 Word analogies (e.g., king:queen :: man:woman) have also been used to evaluate word
 7461 embeddings (Mikolov et al., 2013). In this evaluation, the system is provided with the
 7462 first three parts of the analogy ($i_1 : j_1 :: i_2 : ?$), and the final element is predicted
 7463 by finding the word embedding most similar to $\mathbf{u}_{i_1} - \mathbf{u}_{j_1} + \mathbf{u}_{i_2}$. Another evaluation
 7464 tests whether word embeddings are related to broad lexical semantic categories called
 7465 supersenses (Ciaramita and Johnson, 2003): verbs of motion, nouns that describe animals,
 7466 nouns that describe body parts, and so on. These supersenses are annotated for English
 7467 synsets in WordNet (Fellbaum, 2010). This evaluation is implemented in the qvec metric,
 7468 which tests whether the matrix of supersenses can be reconstructed from the matrix of
 7469 word embeddings (Tsvetkov et al., 2015).

7470 Levy et al. (2015) compared several dense word representations for English — including
 7471 latent semantic analysis, word2vec, and GloVe — using six word similarity metrics and
 7472 two analogy tasks. None of the embeddings outperformed the others on every task,
 7473 but skipgrams were the most broadly competitive. Hyperparameter tuning played a key
 7474 role: any method will perform badly if the wrong hyperparameters are used. Relevant
 7475 hyperparameters include the embedding size, as well as algorithm-specific details such as
 7476 the neighborhood size and the number of negative samples.

7477 14.6.2 Extrinsic evaluations

7478 Word representations contribute to downstream tasks like sequence labeling and document
 7479 classification by enabling generalization across words. The use of distributed representations
 7480 as features is a form of semi-supervised learning, in which performance on a supervised
 7481 learning problem is augmented by learning distributed representations from unlabeled
 7482 data (Miller et al., 2004; Koo et al., 2008; Turian et al., 2010). These pre-trained word
 7483 representations can be used as features in a linear prediction model, or as the input layer
 7484 in a neural network, such as a Bi-LSTM tagging model (§ 7.6). Word representations
 7485 can be evaluated by the performance of the downstream systems that consume them:
 7486 for example, GloVe embeddings are convincingly better than Latent Semantic Analysis
 7487 as features in the downstream task of named entity recognition (Pennington et al., 2014).
 7488 Unfortunately, extrinsic and intrinsic evaluations do not always point in the same direction,

7489 and the best word representations for one downstream task may perform poorly on another
7490 task (Schnabel et al., 2015).

7491 When word representations are updated from labeled data in the downstream task, they
7492 are said to be fine-tuned. When labeled data is plentiful, pre-training may be unnecessary;
7493 when labeled data is scarce, fine-tuning may lead to overfitting. Various combinations of
7494 pre-training and fine-tuning can be employed. Pre-trained embeddings can be used as
7495 initialization before fine-tuning, and this can substantially improve performance (Lample
7496 et al., 2016). Alternatively, both fine-tuned and pre-trained embeddings can be used as
7497 inputs in a single model (Kim, 2014).

7498 In semi-supervised scenarios, pretrained word embeddings can be replaced by “contextualized”
7499 word representations (Peters et al., 2018). These contextualized representations are set
7500 to the hidden states of a deep bi-directional LSTM, which is trained as a bi-directional
7501 language model, motivating the name ELMo (embeddings from language models). Given a
7502 supervised learning problem, the language model generates contextualized representations,
7503 which are then used as the base layer in a task-specific supervised neural network. This
7504 approach yields significant gains over pretrained word embeddings on several tasks, presumably
7505 because the contextualized embeddings use unlabeled data to learn how to integrate linguistic
7506 context into the base layer of the supervised neural network.

7507 14.7 Distributed representations beyond distributional statistics

7508 Distributional word representations can be estimated from huge unlabeled datasets, thereby
7509 covering many words that do not appear in labeled data: for example, GloVe embeddings
7510 are estimated from 800 billion tokens of web data,³ while the largest labeled datasets
7511 for NLP tasks are on the order of millions of tokens. Nonetheless, even a dataset of
7512 hundreds of billions of tokens will not cover every word that may be encountered in the
7513 future. Furthermore, many words will appear only a few times, making their embeddings
7514 unreliable. Many languages exceed English in morphological complexity, and thus have
7515 lower token-to-type ratios. When this problem is coupled with small training corpora, it
7516 becomes especially important to leverage other sources of information beyond distributional
7517 statistics.

7518 14.7.1 Word-internal structure

7519 One solution is to incorporate word-internal structure into word embeddings. Purely
7520 distributional approaches consider words as atomic units, but in fact, many words have
7521 internal structure, so that their meaning can be composed from the representations of

³<http://commoncrawl.org/>

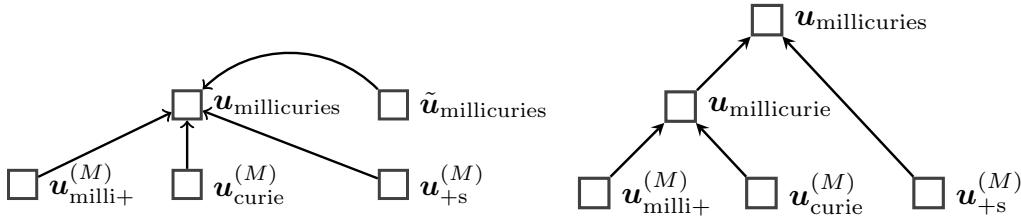


Figure 14.5: Two architectures for building word embeddings from subword units. On the left, morpheme embeddings $\mathbf{u}^{(m)}$ are combined by addition with the non-compositional word embedding $\tilde{\mathbf{u}}$ (Botha and Blunsom, 2014). On the right, morpheme embeddings are combined in a recursive neural network (Luong et al., 2013).

7522 sub-word units. Consider the following terms, all of which are missing from Google’s
 7523 pre-trained word2vec embeddings:⁴

7524 millicuries This word has morphological structure (see § 9.1.2 for more on morphology):
 7525 the prefix milli- indicates an amount, and the suffix -s indicates a plural. (A millicurie
 7526 is an unit of radioactivity.)

7527 caesium This word is a single morpheme, but the characters -ium are often associated with
 7528 chemical elements. (Caesium is the British spelling of a chemical element, spelled
 7529 cesium in American English.)

7530 IAEA This term is an acronym, as suggested by the use of capitalization. The prefix
 7531 I- frequently refers to international organizations, and the suffix -A often refers to
 7532 agencies or associations. (IAEA is the International Atomic Energy Agency.)

7533 Zhezhgan This term is in title case, suggesting the name of a person or place, and the
 7534 character bigram zh indicates that it is likely a transliteration. (Zhezhgan is a mining
 7535 facility in Kazakhstan.)

7536 How can word-internal structure be incorporated into word representations? One
 7537 approach is to construct word representations from embeddings of the characters or morphemes.
 7538 For example, if word i has morphological segments \mathcal{M}_i , then its embedding can be constructed
 7539 by addition (Botha and Blunsom, 2014),

$$\mathbf{u}_i = \tilde{\mathbf{u}}_i + \sum_{j \in \mathcal{M}_i} \mathbf{u}_j^{(M)}, \quad [14.27]$$

7540 where $\mathbf{u}_m^{(M)}$ is a morpheme embedding and $\tilde{\mathbf{u}}_i$ is a non-compositional embedding of the
 7541 whole word, which is an additional free parameter of the model (Figure 14.5, left side). All

⁴<https://code.google.com/archive/p/word2vec/>, accessed September 20, 2017

embeddings are estimated from a log-bilinear language model (Mnih and Hinton, 2007), which is similar to the CBOW model (§ 14.5), but includes only contextual information from preceding words. The morphological segments are obtained using an unsupervised segmenter (Creutz and Lagus, 2007). For words that do not appear in the training data, the embedding can be constructed directly from the morphemes, assuming that each morpheme appears in some other word in the training data. The free parameter \tilde{u} adds flexibility: words with similar morphemes are encouraged to have similar embeddings, but this parameter makes it possible for them to be different.

Word-internal structure can be incorporated into word representations in various other ways. Here are some of the main parameters.

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).

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 model is shown in the right panel of Figure 14.5.

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.

14.7.2 Lexical semantic resources

Resources such as WordNet provide another source of information about word meaning: if we know that caesium is a synonym of cesium, or that a millicurie is a type of measurement unit, then this should help to provide embeddings for the unknown words, and to smooth embeddings of rare words. One way to do this is to retrofit pre-trained word embeddings across a network of lexical semantic relationships (Faruqui et al., 2015) by minimizing the

following objective,

$$\min_{\mathbf{U}} \sum_{j=1}^V \|\mathbf{u}_i - \hat{\mathbf{u}}_i\|_2 + \sum_{(i,j) \in \mathcal{L}} \beta_{ij} \|\mathbf{u}_i - \mathbf{u}_j\|_2, \quad [14.28]$$

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.

14.8 Distributed representations of multiword units

Can distributed representations extend to phrases, sentences, paragraphs, and beyond? Before exploring this possibility, recall the distinction between distributed and distributional representations. Neural embeddings such as word2vec are both distributed (vector-based) and distributional (derived from counts of words in context). As we consider larger units of text, the counts decrease: in the limit, a multi-paragraph span of text would never appear twice, except by plagiarism. Thus, the meaning of a large span of text cannot be determined from distributional statistics alone; it must be computed compositionally from smaller spans. But these considerations are orthogonal to the question of whether distributed representations — dense numerical vectors — are sufficiently expressive to capture the meaning of phrases, sentences, and paragraphs.

14.8.1 Purely distributional methods

Some multiword phrases are non-compositional: the meaning of such phrases is not derived from the meaning of the individual words using typical compositional semantics. This includes proper nouns like San Francisco as well as idiomatic expressions like kick the bucket (Baldwin and Kim, 2010). For these cases, purely distributional approaches can work. A simple approach is to identify multiword units that appear together frequently, and then treat these units as words, learning embeddings using a technique such as word2vec. The problem of identifying multiword units is sometimes called collocation extraction, and can be approached using metrics such as pointwise mutual information: two-word units are extracted first, and then larger units are extracted. Mikolov et al. (2013) identify such units and then treat them as words when estimating skipgram embeddings, showing that the resulting embeddings perform reasonably on a task of solving phrasal analogies, e.g. New York : New York Times :: Baltimore : Baltimore Sun.

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)

7600 14.8.2 Distributional-compositional hybrids

7601 To move beyond short multiword phrases, composition is necessary. A simple but surprisingly
 7602 powerful approach is to represent a sentence with the average of its word embeddings (Mitchell
 7603 and Lapata, 2010). This can be considered a hybrid of the distributional and compositional
 7604 approaches to semantics: the word embeddings are computed distributionally, and then
 7605 the sentence representation is computed by composition.

7606 The word2vec approach can be stretched considerably further, embedding entire sentences
 7607 using a model similar to skipgrams, in the “skip-thought” model of Kiros et al. (2015). Each
 7608 sentence is encoded into a vector using a recurrent neural network: the encoding of sentence
 7609 t is set to the RNN hidden state at its final token, $\mathbf{h}_{M_t}^{(t)}$. This vector is then a parameter
 7610 in a decoder model that is used to generate the previous and subsequent sentences: the
 7611 decoder is another recurrent neural network, which takes the encoding of the neighboring
 7612 sentence as an additional parameter in its recurrent update. (This encoder-decoder model
 7613 is discussed at length in chapter 18.) The encoder and decoder are trained simultaneously
 7614 from a likelihood-based objective, and the trained encoder can be used to compute a
 7615 distributed representation of any sentence. Skip-thought can also be viewed as a hybrid
 7616 of distributional and compositional approaches: the vector representation of each sentence
 7617 is computed compositionally from the representations of the individual words, but the
 7618 training objective is distributional, based on sentence co-occurrence across a corpus.

7619 Autoencoders are a variant of encoder-decoder models in which the decoder is trained
 7620 to produce the same text that was originally encoded, using only the distributed encoding
 7621 vector (Li et al., 2015). The encoding acts as a bottleneck, so that generalization is
 7622 necessary if the model is to successfully fit the training data. In denoising autoencoders, the
 7623 input is a corrupted version of the original sentence, and the auto-encoder must reconstruct

7624 the uncorrupted original (Vincent et al., 2010; Hill et al., 2016). By interpolating between
 7625 distributed representations of two sentences, $\alpha\mathbf{u}_i + (1 - \alpha)\mathbf{u}_j$, it is possible to generate
 7626 sentences that combine aspects of the two inputs, as shown in Figure 14.6 (Bowman et al.,
 7627 2016).

7628 Autoencoders can also be applied to longer texts, such as paragraphs and documents.
 7629 This enables applications such as question answering, which can be performed by matching
 7630 the encoding of the question with encodings of candidate answers (Miao et al., 2016).

7631 14.8.3 Supervised compositional methods

7632 Given a supervision signal, such as a label describing the sentiment or meaning of a
 7633 sentence, a wide range of compositional methods can be applied to compute a distributed
 7634 representation that then predicts the label. The simplest is to average the embeddings of
 7635 each word in the sentence, and pass this average through a feedforward neural network (Iyyer
 7636 et al., 2015). Convolutional and recurrent neural networks go further, with the ability to
 7637 effectively capturing multiword phenomena such as negation (Kalchbrenner et al., 2014;
 7638 Kim, 2014; Li et al., 2015; Tang et al., 2015). Another approach is to incorporate the
 7639 syntactic structure of the sentence into a recursive neural networks, in which the representation
 7640 for each syntactic constituent is computed from the representations of its children (Socher
 7641 et al., 2012). However, in many cases, recurrent neural networks perform as well or better
 7642 than recursive networks (Li et al., 2015).

7643 Whether convolutional, recurrent, or recursive, a key question is whether supervised
 7644 sentence representations are task-specific, or whether a single supervised sentence representation
 7645 model can yield useful performance on other tasks. Wieting et al. (2015) train a variety
 7646 of sentence embedding models for the task of labeling pairs of sentences as paraphrases.
 7647 They show that the resulting sentence embeddings give good performance for sentiment
 7648 analysis. The Stanford Natural Language Inference corpus classifies sentence pairs as
 7649 entailments (the truth of sentence i implies the truth of sentence j), contradictions (the
 7650 truth of sentence i implies the falsity of sentence j), and neutral (i neither entails nor
 7651 contradicts j). Sentence embeddings trained on this dataset transfer to a wide range of
 7652 classification tasks (Conneau et al., 2017).

7653 14.8.4 Hybrid distributed-symbolic representations

7654 The power of distributed representations is in their generality: the distributed representation
 7655 of a unit of text can serve as a summary of its meaning, and therefore as the input for
 7656 downstream tasks such as classification, matching, and retrieval. For example, distributed
 7657 sentence representations can be used to recognize the paraphrase relationship between
 7658 closely related sentences like the following:

7659 (14.5) Donald thanked Vlad profusely.

7660 (14.6) Donald conveyed to Vlad his profound appreciation.

7661 (14.7) Vlad was showered with gratitude by Donald.

7662 Symbolic representations are relatively brittle to this sort of variation, but are better
7663 suited to describe individual entities, the things that they do, and the things that are done
7664 to them. In examples (14.5)-(14.7), we not only know that somebody thanked someone else,
7665 but we can make a range of inferences about what has happened between the entities named
7666 Donald and Vlad. Because distributed representations do not treat entities symbolically,
7667 they lack the ability to reason about the roles played by entities across a sentence or larger
7668 discourse.⁵ A hybrid between distributed and symbolic representations might give the best
7669 of both worlds: robustness to the many different ways of describing the same event, plus
7670 the expressiveness to support inferences about entities and the roles that they play.

7671 A “top-down” hybrid approach is to begin with logical semantics (of the sort described
7672 in the previous two chapters), and but replace the predefined lexicon with a set of distributional
7673 word clusters (Poon and Domingos, 2009; Lewis and Steedman, 2013). A “bottom-up”
7674 approach is to add minimal symbolic structure to existing distributed representations, such
7675 as vector representations for each entity (Ji and Eisenstein, 2015; Wiseman et al., 2016).
7676 This has been shown to improve performance on two problems that we will encounter in the
7677 following chapters: classification of discourse relations between adjacent sentences (chapter 16;
7678 Ji and Eisenstein, 2015), and coreference resolution of entity mentions (chapter 15; Wiseman
7679 et al., 2016; Ji et al., 2017). Research on hybrid semantic representations is still in an early
7680 stage, and future representations may deviate more boldly from existing symbolic and
7681 distributional approaches.

7682 Additional resources

7683 Turney and Pantel (2010) survey a number of facets of vector word representations, focusing
7684 on matrix factorization methods. Schnabel et al. (2015) highlight problems with similarity-based
7685 evaluations of word embeddings, and present a novel evaluation that controls for word
7686 frequency. Baroni et al. (2014) address linguistic issues that arise in attempts to combine
7687 distributed and compositional representations.

7688 In bilingual and multilingual distributed representations, embeddings are estimated for
7689 translation pairs or tuples, such as (dog, perro, chien). These embeddings can improve
7690 machine translation (Zou et al., 2013; Klementiev et al., 2012), transfer natural language
7691 processing models across languages (Täckström et al., 2012), and make monolingual word
7692 embeddings more accurate (Faruqui and Dyer, 2014). A typical approach is to learn a

⁵At a 2014 workshop on semantic parsing, this critique of distributed representations was expressed by Ray Mooney — a leading researcher in computational semantics — in a now well-known quote, “you can’t cram the meaning of a whole sentence into a single vector!”

7693 projection that maximizes the correlation of the distributed representations of each element
 7694 in a translation pair, which can be obtained from a bilingual dictionary. Distributed
 7695 representations can also be linked to perceptual information, such as image features. Bruni
 7696 et al. (2014) use textual descriptions of images to obtain visual contextual information for
 7697 various words, which supplements traditional distributional context. Image features can
 7698 also be inserted as contextual information in log bilinear language models (Kiros et al.,
 7699 2014), making it possible to automatically generate text descriptions of images.

7700 Exercises

- 7701 1. Prove that the sum of probabilities of paths through a hierarchical softmax tree is
 7702 equal to one.
- 7703 2. In skipgram word embeddings, the negative sampling objective can be written as,

$$\mathcal{L} = \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{C}} \text{count}(i, j) \psi(i, j), \quad [14.29]$$

7703 with $\psi(i, j)$ is defined in Equation 14.23.

7704 Suppose we draw the negative samples from the empirical unigram distribution $\hat{p}(i) =$
 7705 $P_{\text{unigram}}(i)$. First, compute the expectation of \mathcal{L} with respect this probability.

7706 Next, take the derivative of this expectation with respect to the score of a single word
 7707 context pair $\sigma(\mathbf{u}_i \cdot \mathbf{v}_j)$, and solve for the pointwise mutual information $\text{PMI}(i, j)$. You
 7708 should be able to show that at the optimum, the PMI is a simple function of $\sigma(\mathbf{u}_i \cdot \mathbf{v}_j)$
 7709 and the number of negative samples.

- 7710 3. * In Brown clustering, prove that the cluster merge that maximizes the average
 7711 mutual information (Equation 14.13) also maximizes the log-likelihood objective
 7712 (Equation 14.12).
- 7713 4. A simple way to compute a distributed phrase representation is to add up the
 7714 distributed representations of the words in the phrase. Consider a sentiment analysis
 model in which the predicted sentiment is, $\psi(\mathbf{w}) = \boldsymbol{\theta} \cdot (\sum_{m=1}^M \mathbf{x}_m)$, where \mathbf{x}_m is
 the vector representation of word m . Prove that in such a model, the following two
 inequalities cannot both hold:

$$\psi(\text{good}) > \psi(\text{not good}) \quad [14.30]$$

$$\psi(\text{bad}) < \psi(\text{not bad}). \quad [14.31]$$

7713 Then construct a similar example pair for the case in which phrase representations
 7714 are the average of the word representations.

5. Now let's consider a slight modification to the prediction model in the previous problem:

$$\psi(\mathbf{w}) = \boldsymbol{\theta} \cdot \text{ReLU}\left(\sum_{m=1}^M \mathbf{x}_m\right) \quad [14.32]$$

7715 Show that in this case, it is possible to achieve the inequalities above. Your solution
 7716 should provide the weights $\boldsymbol{\theta}$ and the embeddings \mathbf{x}_{good} , \mathbf{x}_{bad} , and \mathbf{x}_{not} .

7717 For the next two problems, download a set of pre-trained word embeddings, such as the
 7718 word2vec or polyglot embeddings.

7719 6. Use cosine similarity to find the most similar words to: dog, whale, before, however,
 7720 fabricate.

7721 7. Use vector addition and subtraction to compute target vectors for the analogies below.
 7722 After computing each target vector, find the top three candidates by cosine similarity.

- 7723 • dog:puppy :: cat: ?
- 7724 • speak:speaker :: sing:?
- 7725 • France:French :: England:?
- 7726 • France:wine :: England:?

7727 The remaining problems will require you to build a classifier and test its properties. Pick
 7728 a multi-class text classification dataset, such as RCV1⁶). Divide your data into training
 7729 (60%), development (20%), and test sets (20%), if no such division already exists.

7730 8. Train a convolutional neural network, with inputs set to pre-trained word embeddings
 7731 from the previous problem. Use a special, fine-tuned embedding for out-of-vocabulary
 7732 words. Train until performance on the development set does not improve. You can
 7733 also use the development set to tune the model architecture, such as the convolution
 7734 width and depth. Report F -measure and accuracy, as well as training time.

7735 9. Now modify your model from the previous problem to fine-tune the word embeddings.
 7736 Report F -measure, accuracy, and training time.

7737 10. Try a simpler approach, in which word embeddings in the document are averaged,
 7738 and then this average is passed through a feed-forward neural network. Again, use
 7739 the development data to tune the model architecture. How close is the accuracy to
 7740 the convolutional networks from the previous problems?

⁶http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm

₇₇₄₁ Chapter 15

₇₇₄₂ Reference Resolution

₇₇₄₃ References are one of the most noticeable forms of linguistic ambiguity, afflicting not just
₇₇₄₄ automated natural language processing systems, but also fluent human readers. Warnings
₇₇₄₅ to avoid “ambiguous pronouns” are ubiquitous in manuals and tutorials on writing style.
₇₇₄₆ But referential ambiguity is not limited to pronouns, as shown in the text in Figure 15.1.
₇₇₄₇ Each of the bracketed substrings refers to an entity that is introduced earlier in the passage.
₇₇₄₈ These references include the pronouns he and his, but also the shortened name Cook, and
₇₇₄₉ nominals such as the firm and the firm’s biggest growth market.

₇₇₅₀ Reference resolution subsumes several subtasks. This chapter will focus on coreference
₇₇₅₁ resolution, which is the task of grouping spans of text that refer to a single underlying
₇₇₅₂ entity, or, in some cases, a single event: for example, the spans Tim Cook, he, and Cook
₇₇₅₃ are all coreferent. These individual spans are called mentions, because they mention an
₇₇₅₄ entity; the entity is sometimes called the referent. Each mention has a set of antecedents,
₇₇₅₅ which are preceding mentions that are coreferent; for the first mention of an entity, the
₇₇₅₆ antecedent set is empty. The task of pronominal anaphora resolution requires identifying
₇₇₅₇ only the antecedents of pronouns. In entity linking, references are resolved not to other
₇₇₅₈ spans of text, but to entities in a knowledge base. This task is discussed in chapter 17.

₇₇₅₉ Coreference resolution is a challenging problem for several reasons. Resolving different
₇₇₆₀ types of referring expressions requires different types of reasoning: the features and methods
₇₇₆₁ that are useful for resolving pronouns are different from those that are useful to resolve
₇₇₆₂ names and nominals. Coreference resolution involves not only linguistic reasoning, but also
₇₇₆₃ world knowledge and pragmatics: you may not have known that China was Apple’s biggest
₇₇₆₄ growth market, but it is likely that you effortlessly resolved this reference while reading
₇₇₆₅ the passage in Figure 15.1.¹ A further challenge is that coreference resolution decisions

¹This interpretation is based in part on the assumption that a cooperative author would not use the expression the firm’s biggest growth market to refer to an entity not yet mentioned in the article (Grice, 1975). Pragmatics is the discipline of linguistics concerned with the formalization of such assumptions (Huang,

- (15.1) [[Apple Inc] Chief Executive Tim Cook] has jetted into [China] for talks with government officials as [he] seeks to clear up a pile of problems in [[the firm] ’s biggest growth market] ... [Cook] is on [his] first trip to [the country] since taking over...
-

Figure 15.1: Running example (Yee and Jones, 2012). Coreferring entity mentions are underlined and bracketed.

7766 are often entangled: each mention adds information about the entity, which affects other
 7767 coreference decisions. This means that coreference resolution must be addressed as a
 7768 structure prediction problem. But as we will see, there is no dynamic program that allows
 7769 the space of coreference decisions to be searched efficiently.

7770 15.1 Forms of referring expressions

7771 There are three main forms of referring expressions — pronouns, names, and nominals.

7772 15.1.1 Pronouns

7773 Pronouns are a closed class of words that are used for references. A natural way to think
 7774 about pronoun resolution is Smash (Kehler, 2007):

- 7775 • Search for candidate antecedents;
 7776 • Match against hard agreement constraints;
 7777 • And Select using Heuristics, which are “soft” constraints such as recency, syntactic
 7778 prominence, and parallelism.

7779 15.1.1.1 Search

7780 In the search step, candidate antecedents are identified from the preceding text or speech.²
 7781 Any noun phrase can be a candidate antecedent, and pronoun resolution usually requires

2015).

²Pronouns whose referents come later are known as cataphora, as in this example from Márquez (1970):

- (15.1) Many years later, as [he] faced the firing squad, [Colonel Aureliano Buendía] was to remember
 that distant afternoon when his father took him to discover ice.

7782 parsing the text to identify all such noun phrases.³ Filtering heuristics can help to prune
 7783 the search space to noun phrases that are likely to be coreferent (Lee et al., 2013; Durrett
 7784 and Klein, 2013). In nested noun phrases, mentions are generally considered to be the
 7785 largest unit with a given head word: thus, Apple Inc. Chief Executive Tim Cook would
 7786 be included as a mention, but Tim Cook would not, since they share the same head word,
 7787 Cook.

7788 15.1.1.2 Matching constraints for pronouns

7789 References and their antecedents must agree on semantic features such as number, person,
 7790 gender, and animacy. Consider the pronoun *he* in this passage from the running example:

7791 (15.2) Tim Cook has jetted in for talks with officials as [he] seeks to clear up a pile of
 7792 problems...

7793 The pronoun and possible antecedents have the following features:

- 7794 • *he*: singular, masculine, animate, third person
- 7795 • *officials*: plural, animate, third person
- 7796 • *talks*: plural, inanimate, third person
- 7797 • Tim Cook: singular, masculine, animate, third person

7798 The Smash method searches backwards from *he*, discarding *officials* and *talks* because they
 7799 do not satisfy the agreements constraints.

7800 Another source of constraints comes from syntax — specifically, from the phrase structure
 7801 trees discussed in chapter 10. Consider a parse tree in which both *x* and *y* are phrasal
 7802 constituents. The constituent *x* c-commands the constituent *y* iff the first branching node
 7803 above *x* also dominates *y*. For example, in Figure 15.2a, Abigail c-commands her, because
 7804 the first branching node above Abigail, S, also dominates her. Now, if *x* c-commands *y*,
 7805 government and binding theory (Chomsky, 1982) states that *y* can refer to *x* only if it is a
 7806 reflexive pronoun (e.g., *herself*). Furthermore, if *y* is a reflexive pronoun, then its antecedent
 7807 must c-command it. Thus, in Figure 15.2a, *her* cannot refer to Abigail; conversely, if we
 7808 replace *her* with *herself*, then the reflexive pronoun must refer to Abigail, since this is the
 7809 only candidate antecedent that c-commands it.

7810 Now consider the example shown in Figure 15.2b. Here, Abigail does not c-command
 7811 her, but Abigail’s mom does. Thus, *her* can refer to Abigail — and we cannot use reflexive

³In the OntoNotes coreference annotations, verbs can also be antecedents, if they are later referenced by nominals (Pradhan et al., 2011):

(15.1) Sales of passenger cars [grew] 22%. [The strong growth] followed year-to-year increases.

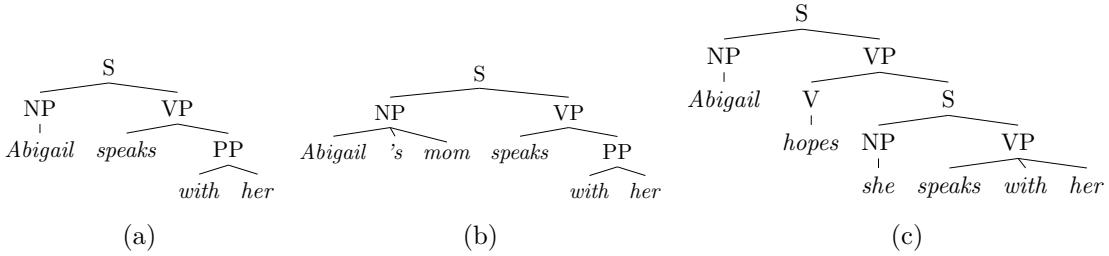


Figure 15.2: In (a), Abigail c-commands her; in (b), Abigail does not c-command her, but Abigail’s mom does; in (c), the scope of Abigail is limited by the S non-terminal, so that she or her can bind to Abigail, but not both.

7812 herself in this context, unless we are talking about Abigail’s mom. However, her does not
 7813 have to refer to Abigail. Finally, Figure 15.2c shows the how these constraints are limited.
 7814 In this case, the pronoun she can refer to Abigail, because the S non-terminal puts Abigail
 7815 outside the domain of she. Similarly, her can also refer to Abigail. But she and her cannot
 7816 be coreferent, because she c-commands her.

7817 15.1.1.3 Heuristics

7818 After applying constraints, heuristics are applied to select among the remaining candidates.
 7819 Recency is a particularly strong heuristic. All things equal, readers will prefer the more
 7820 recent referent for a given pronoun, particularly when comparing referents that occur in
 7821 different sentences. Jurafsky and Martin (2009) offer the following example:

7822 (15.3) The doctor found an old map in the captain’s chest. Jim found an even older
 7823 map hidden on the shelf. [It] described an island.

7824 Readers are expected to prefer the older map as the referent for the pronoun it.

7825 However, subjects are often preferred over objects, and this can contradict the preference
 7826 for recency when two candidate referents are in the same sentence. For example,

7827 (15.4) Asha loaned Mei a book on Spanish. [She] is always trying to help people.

7828 Here, we may prefer to link she to Asha rather than Mei, because of Asha’s position in
 7829 the subject role of the preceding sentence. (Arguably, this preference would not be strong
 7830 enough to select Asha if the second sentence were She is visiting Valencia next month.)

7831 A third heuristic is parallelism:

7832 (15.5) Asha loaned Mei a book on Spanish. Olya loaned [her] a book on Portuguese.

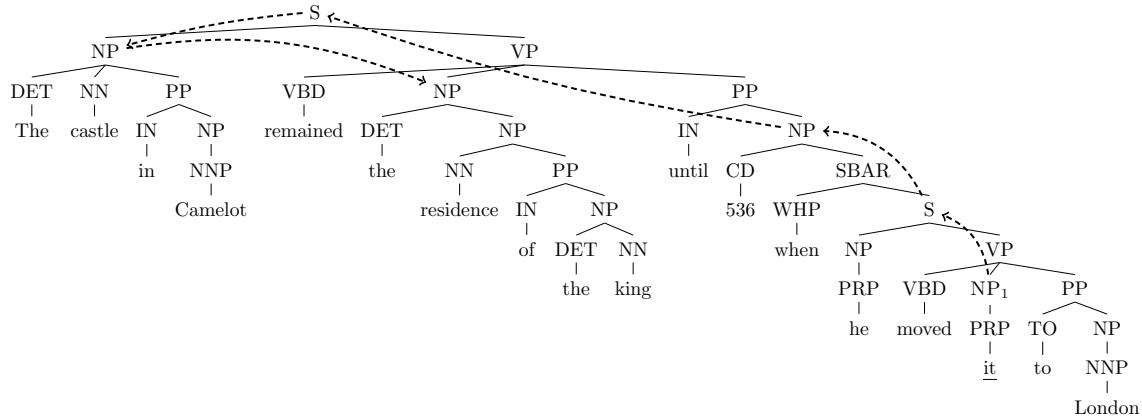


Figure 15.3: Left-to-right breadth-first tree traversal (Hobbs, 1978), indicating that the search for an antecedent for it (NP₁) would proceed in the following order: 536; the castle in Camelot; the residence of the king; Camelot; the king. Hobbs (1978) proposes semantic constraints to eliminate 536 and the castle in Camelot as candidates, since they are unlikely to be the direct object of the verb move.

7833 Here Mei is preferred as the referent for her, contradicting the preference for the subject
 7834 Asha in the preceding sentence.

7835 The recency and subject role heuristics can be unified by traversing the document in
 7836 a syntax-driven fashion (Hobbs, 1978): each preceding sentence is traversed breadth-first,
 7837 left-to-right (Figure 15.3). This heuristic successfully handles (15.4): Asha is preferred as
 7838 the referent for she because the subject NP is visited first. It also handles (15.3): the older
 7839 map is preferred as the referent for it because the more recent sentence is visited first. (An
 7840 alternative unification of recency and syntax is proposed by centering theory (Grosz et al.,
 7841 1995), which is discussed in detail in chapter 16.)

7842 In early work on reference resolution, the number of heuristics was small enough that
 7843 a set of numerical weights could be set by hand (Lappin and Leass, 1994). More recent
 7844 work uses machine learning to quantify the importance of each of these factors. However,
 7845 pronoun resolution cannot be completely solved by constraints and heuristics alone. This
 7846 is shown by the classic example pair (Winograd, 1972):

7847 (15.6) The [city council] denied [the protesters] a permit because [they] advocated/feared
 7848 violence.

7849 Without reasoning about the motivations of the city council and protesters, it is unlikely
 7850 that any system could correctly resolve both versions of this example.

7851 15.1.1.4 Non-referential pronouns

7852 While pronouns are generally used for reference, they need not refer to entities. The
 7853 following examples show how pronouns can refer to propositions, events, and speech acts.

7854 (15.7) They told me that I was too ugly for show business, but I didn't believe [it].

7855 (15.8) Asha saw Babak get angry, and I saw [it] too.

7856 (15.9) Asha said she worked in security. I suppose [that]'s one way to put it.

7857 These forms of reference are generally not annotated in large-scale coreference resolution
 7858 datasets such as OntoNotes (Pradhan et al., 2011).

7859 Pronouns may also have generic referents:

7860 (15.10) A poor carpenter blames [her] tools.

7861 (15.11) On the moon, [you] have to carry [your] own oxygen.

7862 (15.12) Every farmer who owns a donkey beats [it]. (Geach, 1962)

7863 In the OntoNotes dataset, coreference is not annotated for generic referents, even in cases
 7864 like these examples, in which the same generic entity is mentioned multiple times.

7865 Some pronouns do not refer to anything at all:

7866 (15.13) [It]'s raining.

[Il] pleut. (Fr)

7867 (15.14) [It] 's money that she's really after.

7868 (15.15) [It] is too bad that we have to work so hard.

7869 How can we automatically distinguish these usages of it from referential pronouns?
 7870 Consider the the difference between the following two examples (Bergsma et al., 2008):

7871 (15.16) You can make [it] in advance.

7872 (15.17) You can make [it] in showbiz.

7873 In the second example, the pronoun it is non-referential. One way to see this is by
 7874 substituting another pronoun, like them, into these examples:

7875 (15.18) You can make [them] in advance.

7876 (15.19) ? You can make [them] in showbiz.

7877 The questionable grammaticality of the second example suggests that it is not referential.
 7878 Bergsma et al. (2008) operationalize this idea by comparing distributional statistics for the

7879 *n*-grams around the word it, testing how often other pronouns or nouns appear in the same
7880 context. In cases where nouns and other pronouns are infrequent, the it is unlikely to be
7881 referential.

7882 15.1.2 Proper Nouns

7883 If a proper noun is used as a referring expression, it often corefers with another proper
7884 noun, so that the coreference problem is simply to determine whether the two names match.
7885 Subsequent proper noun references often use a shortened form, as in the running example
7886 (Figure 15.1):

7887 (15.20) Apple Inc Chief Executive [Tim Cook] has jetted into China ...[Cook] is on his
7888 first business trip to the country ...

7889 A typical solution for proper noun coreference is to match the syntactic head words of
7890 the reference with the referent. In § 10.5.2, we saw that the head word of a phrase can
7891 be identified by applying head percolation rules to the phrasal parse tree; alternatively,
7892 the head can be identified as the root of the dependency subtree covering the name. For
7893 sequences of proper nouns, the head word will be the final token.

7894 There are a number of caveats to the practice of matching head words of proper nouns.

- 7895 • In the European tradition, family names tend to be more specific than given names,
7896 and family names usually come last. However, other traditions have other practices:
7897 for example, in Chinese names, the family name typically comes first; in Japanese,
7898 honorifics come after the name, as in Nobu-San (Mr. Nobu).
- 7899 • In organization names, the head word is often not the most informative, as in Georgia
7900 Tech and Virginia Tech. Similarly, Lebanon does not refer to the same entity as
7901 Southern Lebanon, necessitating special rules for the specific case of geographical
7902 modifiers (Lee et al., 2011).
- 7903 • Proper nouns can be nested, as in [the CEO of [Microsoft]], resulting in head word
7904 match without coreference.

7905 Despite these difficulties, proper nouns are the easiest category of references to resolve (Stoyanov
7906 et al., 2009). In machine learning systems, one solution is to include a range of matching
7907 features, including exact match, head match, and string inclusion. In addition to matching
7908 features, competitive systems (e.g., Bengtson and Roth, 2008) include large lists, or gazetteers,
7909 of acronyms (e.g., the National Basketball Association/NBA), demonymns (e.g., the Israelis/Israel),
7910 and other aliases (e.g., the Georgia Institute of Technology/Georgia Tech).

7911 15.1.3 Nominals

7912 In coreference resolution, noun phrases that are neither pronouns nor proper nouns are
 7913 referred to as nominals. In the running example (Figure 15.1), nominal references include:
 7914 the firm (Apple Inc); the firm’s biggest growth market (China); and the country (China).

7915 Nominals are especially difficult to resolve (Denis and Baldridge, 2007; Durrett and
 7916 Klein, 2013), and the examples above suggest why this may be the case: world knowledge
 7917 is required to identify Apple Inc as a firm, and China as a growth market. Other difficult
 7918 examples include the use of colloquial expressions, such as coreference between Clinton
 7919 campaign officials and the Clinton camp (Soon et al., 2001).

7920 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:

$$c_1 = \{\text{Apple Inc}_{1:2}, \text{the firm}_{27:28}\} \quad [15.1]$$

$$c_2 = \{\text{Apple Inc Chief Executive Tim Cook}_{1:6}, \text{he}_{17}, \text{Cook}_{33}, \text{his}_{36}\} \quad [15.2]$$

$$c_3 = \{\text{China}_{10}, \text{the firm's biggest growth market}_{27:32}, \text{the country}_{40:41}\} \quad [15.3]$$

7921 Each row specifies the token spans that mention an entity. (“Singleton” entities, which are
 7922 mentioned only once (e.g., talks, government officials), are excluded from the annotations.)
 7923 Equivalently, if given a set of M mentions, $\{m_i\}_{i=1}^M$, each mention i can be assigned to a
 7924 cluster z_i , where $z_i = z_j$ if i and j are coreferent. The cluster assignments z are invariant
 7925 under permutation. The unique clustering associated with the assignment z is written
 7926 $c(z)$.

7927 Mention identification The task of identifying mention spans for coreference resolution
 7928 is often performed by applying a set of heuristics to the phrase structure parse of each
 7929 sentence. A typical approach is to start with all noun phrases and named entities, and
 7930 then apply filtering rules to remove nested noun phrases with the same head (e.g., [Apple
 7931 CEO [Tim Cook]]), numeric entities (e.g., [100 miles], [97%]), non-referential it, etc (Lee
 7932 et al., 2013; Durrett and Klein, 2013). In general, these deterministic approaches err in
 7933 favor of recall, since the mention clustering component can choose to ignore false positive
 7934 mentions, but cannot recover from false negatives. An alternative is to consider all spans

⁴In many annotations, the term markable is used to refer to spans of text that can potentially mention an entity. The set of markables includes non-referential pronouns, which does not mention any entity. Part of the job of the coreference system is to avoid incorrectly linking these non-referential markables to any mention chains.

7935 (up to some finite length) as candidate mentions, performing mention identification and
 7936 clustering jointly (Daumé III and Marcu, 2005; Lee et al., 2017).

7937 Mention clustering The overwhelming majority of research on coreference resolution addresses
 7938 the subtask of mention clustering, and this will be the focus of the remainder of this chapter.
 7939 There are two main sets of approaches. In mention-based models, the scoring function for
 7940 a coreference clustering decomposes over pairs of mentions. These pairwise decisions are
 7941 then aggregated, using a clustering heuristic. Mention-based coreference clustering can be
 7942 treated as a fairly direct application of supervised classification or ranking. However, the
 7943 mention-pair locality assumption can result in incoherent clusters, like {Hillary Clinton ←
 7944 Clinton ← Mr Clinton}, in which the pairwise links score well, but the overall result
 7945 is unsatisfactory. Entity-based models address this issue by scoring entities holistically.
 7946 This can make inference more difficult, since the number of possible entity groupings is
 7947 exponential in the number of mentions.

7948 15.2.1 Mention-pair models

7949 In the mention-pair model, a binary label $y_{i,j} \in \{0, 1\}$ is assigned to each pair of mentions
 7950 (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
 7951 mention he in Figure 15.1 is preceded by five other mentions: (1) Apple Inc; (2) Apple
 7952 Inc Chief Executive Tim Cook; (3) China; (4) talks; (5) government officials. The correct
 7953 mention pair labeling is $y_{2,6} = 1$ and $y_{i \neq 2,6} = 0$ for all other i . If a mention j introduces a
 7954 new entity, such as mention 3 in the example, then $y_{i,j} = 0$ for all i . The same is true for
 7955 “mentions” that do not refer to any entity, such as non-referential pronouns. If mention j
 7956 refers to an entity that has been mentioned more than once, then $y_{i,j} = 1$ for all $i < j$ that
 7957 mention the referent.

7958 By transforming coreference into a set of binary labeling problems, the mention-pair
 7959 model makes it possible to apply an off-the-shelf binary classifier (Soon et al., 2001). This
 7960 classifier is applied to each mention j independently, searching backwards from j until
 7961 finding an antecedent i which corefers with j with high confidence. After identifying a
 7962 single antecedent, the remaining mention pair labels can be computed by transitivity: if
 7963 $y_{i,j} = 1$ and $y_{j,k} = 1$, then $y_{i,k} = 1$.

7964 Since the ground truth annotations give entity chains \mathbf{c} but not individual mention-pair
 7965 labels \mathbf{y} , an additional heuristic must be employed to convert the labeled data into training
 7966 examples for classification. A typical approach is to generate at most one positive labeled
 7967 instance $y_{a_j,j} = 1$ for mention j , where a_j is the index of the most recent antecedent,
 7968 $a_j = \max\{i : i < j \wedge z_i = z_j\}$. Negative labeled instances are generated for all for all
 7969 $i \in \{a_j + 1, \dots, j\}$. In the running example, the most recent antecedent of the pronoun he
 7970 is $a_6 = 2$, so the training data would be $y_{2,6} = 1$ and $y_{3,6} = y_{4,6} = y_{5,6} = 0$. The variable

7971 $y_{1,6}$ is not part of the training data, because the first mention appears before the true
 7972 antecedent $a_6 = 2$.

7973 15.2.2 Mention-ranking models

In mention ranking (Denis and Baldridge, 2007), the classifier learns to identify a single antecedent $a_i \in \{\epsilon, 1, 2, \dots, i-1\}$ for each referring expression i ,

$$\hat{a}_i = \operatorname{argmax}_{a \in \{\epsilon, 1, 2, \dots, i-1\}} \psi_M(a, i), \quad [15.4]$$

7974 where $\psi_M(a, i)$ is a score for the mention pair (a, i) . If $a = \epsilon$, then mention i does not
 7975 refer to any previously-introduced entity — it is not anaphoric. Mention-ranking is similar
 7976 to the mention-pair model, but all candidates are considered simultaneously, and at most
 7977 a single antecedent is selected. The mention-ranking model explicitly accounts for the
 7978 possibility that mention i is not anaphoric, through the score $\psi_M(\epsilon, i)$. The determination
 7979 of anaphoricity can be made by a special classifier in a preprocessing step, so that non- ϵ
 7980 antecedents are identified only for spans that are determined to be anaphoric (Denis and
 7981 Baldridge, 2008).

7982 As a learning problem, ranking can be trained using the same objectives as in discriminative
 7983 classification. For each mention i , we can define a gold antecedent a_i^* , and an associated
 7984 loss, such as the hinge loss, $\ell_i = (1 - \psi_M(a_i^*, i) + \psi_M(\hat{a}, i))_+$ or the negative log-likelihood,
 7985 $\ell_i = -\log p(a_i^* \mid i; \boldsymbol{\theta})$. (For more on learning to rank, see § 17.1.1.) But as with
 7986 the mention-pair model, there is a mismatch between the labeled data, which comes in
 7987 the form of mention sets, and the desired supervision, which would indicate the specific
 7988 antecedent of each mention. The antecedent variables $\{a_i\}_{i=1}^M$ relate to the mention sets in
 7989 a many-to-one mapping: each set of antecedents induces a single clustering, but a clustering
 7990 can correspond to many different settings of antecedent variables.

A heuristic solution is to set $a_i^* = \max\{j : j < i \wedge z_j = z_i\}$, the most recent mention in
 the same cluster as i . But the most recent mention may not be the most informative: in
 the running example, the most recent antecedent of the mention Cook is the pronoun he,
 but a more useful antecedent is the earlier mention Apple Inc Chief Executive Tim Cook.
 Rather than selecting a specific antecedent to train on, the antecedent can be treated as a
 latent variable, in the manner of the latent variable perceptron from § 12.4.2 (Fernandes

et al., 2014):

$$\hat{\mathbf{a}} = \operatorname{argmax}_{\mathbf{a}} \sum_{i=1}^M \psi_M(a_i, i) \quad [15.5]$$

$$\mathbf{a}^* = \operatorname{argmax}_{\mathbf{a} \in \mathcal{A}(\mathbf{c})} \sum_{i=1}^M \psi_M(a_i, i) \quad [15.6]$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \sum_{i=1}^M \frac{\partial L}{\partial \boldsymbol{\theta}} \psi_M(a_i^*, i) - \sum_{i=1}^M \frac{\partial L}{\partial \boldsymbol{\theta}} \psi_M(\hat{a}_i, i) \quad [15.7]$$

where $\mathcal{A}(\mathbf{c})$ is the set of antecedent structures that is compatible with the ground truth coreference clustering \mathbf{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:

$$p(\mathbf{c} | \mathbf{m}) = \sum_{\mathbf{a} \in \mathcal{A}(\mathbf{c})} p(\mathbf{a} | \mathbf{m}) = \sum_{\mathbf{a} \in \mathcal{A}(\mathbf{c})} \prod_{i=1}^M p(a_i | i, \mathbf{m}) \quad [15.8]$$

$$p(a_i | i, \mathbf{m}) = \frac{\exp(\psi_M(a_i, i))}{\sum_{a' \in \{\epsilon, 1, 2, \dots, i-1\}} \exp(\psi_M(a', i))}. \quad [15.9]$$

7991 This objective rewards models that assign high scores for all valid antecedent structures. In
 7992 the running example, this would correspond to summing the probabilities of the two valid
 7993 antecedents for Cook, he and Apple Inc Chief Executive Tim Cook. In one of the exercises,
 7994 you will compute the number of valid antecedent structures for a given clustering.

7995 15.2.3 Transitive closure in mention-based models

A problem for mention-based models is that individual mention-level decisions may be incoherent. Consider the following mentions:

$$m_1 = \text{Hillary Clinton} \quad [15.10]$$

$$m_2 = \text{Clinton} \quad [15.11]$$

$$m_3 = \text{Bill Clinton} \quad [15.12]$$

7996 A mention-pair system might predict $\hat{y}_{1,2} = 1, \hat{y}_{2,3} = 1, \hat{y}_{1,3} = 0$. Similarly, a mention-ranking
 7997 system might choose $\hat{a}_2 = 1$ and $\hat{a}_3 = 2$. Logically, if mentions 1 and 3 are both coreferent
 7998 with mention 2, then all three mentions must refer to the same entity. This constraint is
 7999 known as transitive closure.

8000 Transitive closure can be applied post hoc, revising the independent mention-pair or
 8001 mention-ranking decisions. However, there are many possible ways to enforce transitive

closure: in the example above, we could set $\hat{y}_{1,3} = 1$, or $\hat{y}_{1,2} = 0$, or $\hat{y}_{2,3} = 0$. For documents with many mentions, there may be many violations of transitive closure, and many possible fixes. Transitive closure can be enforced by always adding edges, so that $\hat{y}_{1,3} = 1$ is preferred (e.g., Soon et al., 2001), but this can result in overclustering, with too many mentions grouped into too few entities.

Mention-pair coreference resolution can be viewed as a constrained optimization problem,

$$\begin{aligned} \max_{\mathbf{y} \in \{0,1\}^M} \quad & \sum_{j=1}^M \sum_{i=1}^j \psi_M(i, j) \times y_{i,j} \\ \text{s.t.} \quad & y_{i,j} + y_{j,k} - 1 \leq y_{i,k}, \quad \forall i < j < k, \end{aligned}$$

with the constraint enforcing transitive closure. This constrained optimization problem is equivalent to graph partitioning with positive and negative edge weights: construct a graph where the nodes are mentions, and the edges are the pairwise scores $\psi_M(i, j)$; the goal is to partition the graph so as to maximize the sum of the edge weights between all nodes within the same partition (McCallum and Wellner, 2004). This problem is NP-hard, motivating approximations such as correlation clustering (Bansal et al., 2004) and integer linear programming (Klenner, 2007; Finkel and Manning, 2008, also see § 13.2.2).

15.2.4 Entity-based models

A weakness of mention-based models is that they treat coreference resolution as a classification or ranking problem, when it is really a clustering problem: the goal is to group the mentions together into clusters that correspond to the underlying entities. Entity-based approaches attempt to identify these clusters directly. Such methods require a scoring function at the entity level, measuring whether each set of mentions is internally consistent. Coreference resolution can then be viewed as the following optimization,

$$\max_{\mathbf{z}} \quad \sum_{e=1} \psi_E(\{i : z_i = e\}), \tag{15.13}$$

where z_i indicates the entity referenced by mention i , and $\psi_E(\{i : z_i = e\})$ is a scoring function applied to all mentions i that are assigned to entity e .

Entity-based coreference resolution is conceptually similar to the unsupervised clustering problems encountered in chapter 5: the goal is to obtain clusters of mentions that are internally coherent. The number of possible clusterings is the Bell number, which is defined by the following recurrence (Bell, 1934; Luo et al., 2004),

$$B_n = \sum_{k=0}^{n-1} B_k \binom{n-1}{k} = \frac{1}{e} \sum_{k=0}^{\infty} \frac{k^n}{k!}. \tag{15.14}$$

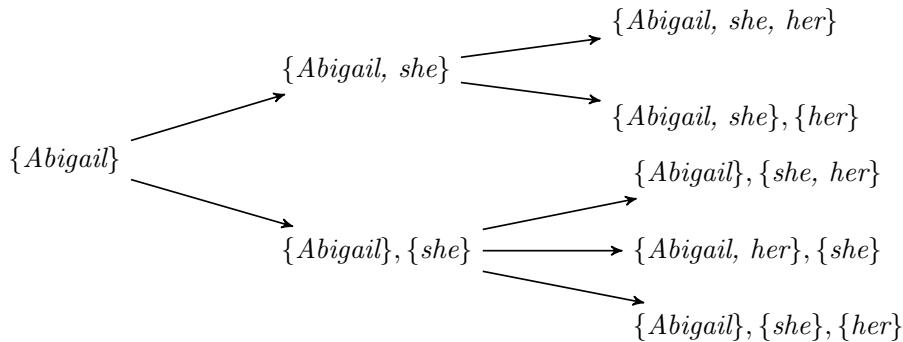


Figure 15.4: The Bell Tree for the sentence Abigail hopes she speaks with her. Which paths are excluded by the syntactic constraints mentioned in § 15.1.1?

8017 This recurrence is illustrated by the Bell tree, which is applied to a short coreference
 8018 problem in Figure 15.4. The Bell number B_n grows exponentially with n , making exhaustive
 8019 search of the space of clusterings impossible. For this reason, entity-based coreference
 8020 resolution typically involves incremental search, in which clustering decisions are based on
 8021 local evidence, in the hope of approximately optimizing the full objective in Equation 15.13.
 8022 This approach is sometimes called cluster ranking, in contrast to mention ranking.

8023 *Generative models of coreference Entity-based cooreference can be approached through
 8024 probabilistic generative models, in which the mentions in the document are conditioned on
 8025 a set of latent entities (Haghghi and Klein, 2007, 2010). An advantage of these methods
 8026 is that they can be learned from unlabeled data (Poon and Domingos, 2008, e.g.,); a
 8027 disadvantage is that probabilistic inference is required not just for learning, but also for
 8028 prediction. Furthermore, generative models require independence assumptions that are
 8029 difficult to apply in coreference resolution, where the diverse and heterogeneous features
 8030 do not admit an easy decomposition into mutually independent subsets.

8031 15.2.4.1 Incremental cluster ranking

8032 The Smash method (§ 15.1.1) can be extended to entity-based coreference resolution by
 8033 building up coreference clusters while moving through the document (Cardie and Wagstaff,
 8034 1999). At each mention, the algorithm iterates backwards through possible antecedent
 8035 clusters; but unlike Smash, a cluster is selected only if all members of its cluster are
 8036 compatible with the current mention. As mentions are added to a cluster, so are their
 8037 features (e.g., gender, number, animacy). In this way, incoherent chains like {Hillary Clinton, Clinton, Bill Clinton}
 8038 can be avoided. However, an incorrect assignment early in the document — a search error
 8039 — might lead to a cascade of errors later on.

More sophisticated search strategies can help to ameliorate the risk of search errors. One approach is beam search (§ 11.3), in which a set of hypotheses is maintained throughout search. Each hypothesis represents a path through the Bell tree (Figure 15.4). Hypotheses are “expanded” either by adding the next mention to an existing cluster, or by starting a new cluster. Each expansion receives a score, based on Equation 15.13, and the top K hypotheses are kept on the beam as the algorithm moves to the next step.

Incremental cluster ranking can be made more accurate by performing multiple passes over the document, applying rules (or “sieves”) with increasing recall and decreasing precision at each pass (Lee et al., 2013). In the early passes, coreference links are proposed only between mentions that are highly likely to corefer (e.g., exact string match for full names and nominals). Information can then be shared among these mentions, so that when more permissive matching rules are applied later, agreement is preserved across the entire cluster. For example, in the case of {Hillary Clinton, Clinton, she}, the name-matching sieve would link Clinton and Hillary Clinton, and the pronoun-matching sieve would then link she to the combined cluster. A deterministic multi-pass system won nearly every track of the 2011 CoNLL shared task on coreference resolution (Pradhan et al., 2011). Given the dominance of machine learning in virtually all other areas of natural language processing — and more than fifteen years of prior work on machine learning for coreference — this was a surprising result, even if learning-based methods have subsequently regained the upper hand (e.g., Lee et al., 2017, the state-of-the-art at the time of this writing).

15.2.4.2 Incremental perceptron

Incremental coreference resolution can be learned with the incremental perceptron, as described in § 11.3.2. At mention i , each hypothesis on the beam corresponds to a clustering of mentions $1 \dots i - 1$, or equivalently, a path through the Bell tree up to position $i - 1$. As soon as none of the hypotheses on the beam are compatible with the gold coreference clustering, a perceptron update is made (Daumé III and Marcu, 2005). For concreteness, consider a linear cluster ranking model,

$$\psi_E(\{i : z_i = e\}) = \sum_{i:z_i=e} \boldsymbol{\theta} \cdot \mathbf{f}(i, \{j : j < i \wedge z_j = e\}), \quad [15.15]$$

where the score for each cluster is computed as the sum of scores of all mentions that are linked into the cluster, and $\mathbf{f}(i, \emptyset)$ is a set of features for the non-anaphoric mention that initiates the cluster.

Using Figure 15.4 as an example, suppose that the ground truth is,

$$\mathbf{c}^* = \{\text{Abigail, her}\}, \{\text{she}\}, \quad [15.16]$$

but that with a beam of size one, the learner reaches the hypothesis,

$$\hat{\mathbf{c}} = \{\text{Abigail, she}\}. \quad [15.17]$$

This hypothesis is incompatible with \mathbf{c}^* , so an update is needed:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{f}(\mathbf{c}^*) - \mathbf{f}(\hat{\mathbf{c}}) \quad [15.18]$$

$$= \boldsymbol{\theta} + (\mathbf{f}(\text{Abigail}, \emptyset) + \mathbf{f}(\text{she}, \emptyset)) - (\mathbf{f}(\text{Abigail}, \emptyset) + \mathbf{f}(\text{she}, \{\text{Abigail}\})) \quad [15.19]$$

$$= \boldsymbol{\theta} + \mathbf{f}(\text{she}, \emptyset) - \mathbf{f}(\text{she}, \{\text{Abigail}\}). \quad [15.20]$$

8066 This style of incremental update can also be applied to a margin loss between the gold
 8067 clustering and the top clustering on the beam. By backpropagating from this loss, it is also
 8068 possible to train a more complicated scoring function, such as a neural network in which
 8069 the score for each entity is a function of embeddings for the entity mentions (Wiseman
 8070 et al., 2015).

8071 15.2.4.3 Reinforcement learning

8072 Reinforcement learning is a topic worthy of a textbook of its own (Sutton and Barto,
 8073 1998),⁵ so this section will provide only a very brief overview, in the context of coreference
 8074 resolution. A stochastic policy assigns a probability to each possible action, conditional
 8075 on the context. The goal is to learn a policy that achieves a high expected reward, or
 8076 equivalently, a low expected cost.

8077 In incremental cluster ranking, a complete clustering on M mentions can be produced
 8078 by a sequence of M actions, in which the action z_i either merges mention i with an existing
 8079 cluster or begins a new cluster. We can therefore create a stochastic policy using the cluster
 8080 scores (Clark and Manning, 2016),

$$\Pr(z_i = e; \boldsymbol{\theta}) = \frac{\exp \psi_E(i \cup \{j : z_j = e\}; \boldsymbol{\theta})}{\sum_{e'} \exp \psi_E(i \cup \{j : z_j = e'\}; \boldsymbol{\theta})}, \quad [15.21]$$

8081 where $\psi_E(i \cup \{j : z_j = e\}; \boldsymbol{\theta})$ is the score under parameters $\boldsymbol{\theta}$ for assigning mention i to
 8082 cluster e . This score can be an arbitrary function of the mention i , the cluster e and its
 8083 (possibly empty) set of mentions; it can also include the history of actions taken thus far.

8084 If a policy assigns probability $p(\mathbf{c}; \boldsymbol{\theta})$ to clustering \mathbf{c} , then its expected loss is,

$$L(\boldsymbol{\theta}) = \sum_{\mathbf{c} \in \mathcal{C}(\mathbf{m})} p_{\boldsymbol{\theta}}(\mathbf{c}) \times \ell(\mathbf{c}), \quad [15.22]$$

8085 where $\mathcal{C}(\mathbf{m})$ is the set of possible clusterings for mentions \mathbf{m} . The loss $\ell(\mathbf{c})$ can be based on
 8086 any arbitrary scoring function, including the complex evaluation metrics used in coreference
 8087 resolution (see § 15.4). This is an advantage of reinforcement learning, which can be trained

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).

8088 directly on the evaluation metric — unlike traditional supervised learning, which requires
 8089 a loss function that is differentiable and decomposable across individual decisions.

Rather than summing over the exponentially many possible clusterings, we can approximate the expectation by sampling trajectories of actions, $\mathbf{z} = (z_1, z_2, \dots, z_M)$, 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 \mathbf{z} corresponds to a single clustering \mathbf{c} , and so we can write the loss of an action sequence as $\ell(\mathbf{c}(\mathbf{z}))$. The policy gradient algorithm computes the gradient of the expected loss as an expectation over trajectories (Sutton et al., 2000),

$$\frac{\partial}{\partial \boldsymbol{\theta}} L(\boldsymbol{\theta}) = E_{\mathbf{z} \sim \mathcal{Z}(\boldsymbol{m})} \ell(\mathbf{c}(\mathbf{z})) \sum_{i=1}^M \frac{\partial}{\partial \boldsymbol{\theta}} \log p(z_i | \mathbf{z}_{1:i-1}, \boldsymbol{m}) \quad [15.23]$$

$$\approx \frac{1}{K} \sum_{k=1}^K \ell(\mathbf{c}(\mathbf{z}^{(k)})) \sum_{i=1}^M \frac{\partial}{\partial \boldsymbol{\theta}} \log p(z_i^{(k)} | \mathbf{z}_{1:i-1}^{(k)}, \boldsymbol{m}) \quad [15.24]$$

[15.25]

8090 where the action sequence $\mathbf{z}^{(k)}$ is sampled from the current policy. Unlike the incremental
 8091 perceptron, an update is not made until the complete action sequence is available.

8092 15.2.4.4 Learning to search

8093 Policy gradient can suffer from high variance: while the average loss over K samples is
 8094 asymptotically equal to the expected reward of a given policy, this estimate may not be
 8095 accurate unless K is very large. This can make it difficult to allocate credit and blame to
 8096 individual actions. In learning to search, this problem is addressed through the addition
 8097 of an oracle policy, which is known to receive zero or small loss. The oracle policy can be
 8098 used in two ways:

- 8099 • The oracle can be used to generate partial hypotheses that are likely to score well,
 8100 by generating i actions from the initial state. These partial hypotheses are then used
 8101 as starting points for the learned policy. This is known as roll-in.
- 8102 • The oracle can be used to compute the minimum possible loss from a given state, by
 8103 generating $M - i$ actions from the current state until completion. This is known as
 8104 roll-out.

8105 The oracle can be combined with the existing policy during both roll-in and roll-out,
 8106 sampling actions from each policy (Daumé III et al., 2009). One approach is to gradually
 8107 decrease the number of actions drawn from the oracle over the course of learning (Ross
 8108 et al., 2011).

Algorithm 18 Learning to search for entity-based coreference resolution

```

1: procedure Compute-gradient(mentions  $\mathbf{m}$ , loss function  $\ell$ , parameters  $\boldsymbol{\theta}$ )
2:    $L(\boldsymbol{\theta}) \leftarrow 0$ 
3:    $\mathbf{z} \sim p(\mathbf{z} | \mathbf{m}; \boldsymbol{\theta})$                                  $\triangleright$  Sample a trajectory from the current policy
4:   for  $i \in \{1, 2, \dots, M\}$  do
5:     for action  $z \in \mathcal{Z}(\mathbf{z}_{1:i-1}, \mathbf{m})$  do       $\triangleright$  All possible actions after history  $\mathbf{z}_{1:i-1}$ 
6:        $\mathbf{h} \leftarrow \mathbf{z}_{1:i-1} \oplus z$                    $\triangleright$  Concatenate history  $\mathbf{z}_{1:i-1}$  with action  $z$ 
7:       for  $j \in \{i+1, i+2, \dots, M\}$  do           $\triangleright$  Roll-out
8:          $h_j \leftarrow \operatorname{argmin}_h \ell(\mathbf{h}_{1:j-1} \oplus h)$      $\triangleright$  Oracle selects action with minimum loss
9:        $L(\boldsymbol{\theta}) \leftarrow L(\boldsymbol{\theta}) + p(z | \mathbf{z}_{1:i-1}, \mathbf{m}; \boldsymbol{\theta}) \times \ell(\mathbf{h})$        $\triangleright$  Update expected loss
10:  return  $\frac{\partial}{\partial \boldsymbol{\theta}} L(\boldsymbol{\theta})$ 

```

8109 In the context of entity-based coreference resolution, Clark and Manning (2016) use
 8110 the learned policy for roll-in and the oracle policy for roll-out. Algorithm 18 shows how
 8111 the gradients on the policy weights are computed in this case. In this application, the
 8112 oracle is “noisy”, because it selects the action that minimizes only the local loss — the
 8113 accuracy of the coreference clustering up to mention i — rather than identifying the action
 8114 sequence that will lead to the best final coreference clustering on the entire document.
 8115 When learning from noisy oracles, it can be helpful to mix in actions from the current
 8116 policy with the oracle during roll-out (Chang et al., 2015).

8117 15.3 Representations for coreference resolution

8118 Historically, coreference resolution has employed an array of hand-engineered features
 8119 to capture the linguistic constraints and preferences described in § 15.1 (Soon et al.,
 8120 2001). Later work has documented the utility of lexical and blexical features on mention
 8121 pairs (Björkelund and Nugues, 2011; Durrett and Klein, 2013). The most recent and
 8122 successful methods replace many (but not all) of these features with distributed representations
 8123 of mentions and entities (Wiseman et al., 2015; Clark and Manning, 2016; Lee et al., 2017).

8124 15.3.1 Features

8125 Coreference features generally rely on a preprocessing pipeline to provide part-of-speech
 8126 tags and phrase structure parses. This pipeline makes it possible to design features that
 8127 capture many of the phenomena from § 15.1, and is also necessary for typical approaches to
 8128 mention identification. However, the pipeline may introduce errors that propagate to the
 8129 downstream coreference clustering system. Furthermore, the existence of such a pipeline

8130 presupposes resources such as treebanks, which do not exist for many languages.⁶

8131 15.3.1.1 Mention features

8132 Features of individual mentions can help to predict anaphoricity. In systems where mention
 8133 detection is performed jointly with coreference resolution, these features can also predict
 8134 whether a span of text is likely to be a mention. For mention i , typical features include:

8135 Mention type. Each span can be identified as a pronoun, name, or nominal, using the
 8136 part-of-speech of the head word of the mention: both the Penn Treebank and Universal
 8137 Dependencies tagsets (§ 8.1.1) include tags for pronouns and proper nouns, and all
 8138 other heads can be marked as nominals (Haghghi and Klein, 2009).

8139 Mention width. The number of tokens in a mention is a rough predictor of its anaphoricity,
 8140 with longer mentions being less likely to refer back to previously-defined entities.

8141 Lexical features. The first, last, and head words can help to predict anaphoricity; they
 8142 are also useful in conjunction with features such as mention type and part-of-speech,
 8143 providing a rough measure of agreement (Björkelund and Nugues, 2011). The number
 8144 of lexical features can be very large, so it can be helpful to select only frequently-occurring
 8145 features (Durrett and Klein, 2013).

8146 Morphosyntactic features. These features include the part-of-speech, number, gender, and
 8147 dependency ancestors.

8148 The features for mention i and candidate antecedent a can be conjoined, producing
 8149 joint features that can help to assess the compatibility of the two mentions. For example,
 8150 Durrett and Klein (2013) conjoin each feature with the mention types of the anaphora
 8151 and the antecedent. Coreference resolution corpora such as ACE and OntoNotes contain
 8152 documents from various genres. By conjoining the genre with other features, it is possible
 8153 to learn genre-specific feature weights.

8154 15.3.1.2 Mention-pair features

8155 For any pair of mentions i and j , typical features include:

8156 Distance. The number of intervening tokens, mentions, and sentences between i and j can
 8157 all be used as distance features. These distances can be computed on the surface
 8158 text, or on a transformed representation reflecting the breadth-first tree traversal

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>

8159 (Figure 15.3). Rather than using the distances directly, they are typically binned,
8160 creating binary features.

8161 String match. A variety of string match features can be employed: exact match, suffix
8162 match, head match, and more complex matching rules that disregard irrelevant
8163 modifiers (Soon et al., 2001).

8164 Compatibility. Building on the model, features can measure the anaphor and antecedent
8165 agree with respect to morphosyntactic attributes such as gender, number, and animacy.

8166 Nesting. If one mention is nested inside another (e.g., [The President of [France]]), they
8167 generally cannot corefer.

8168 Same speaker. For documents with quotations, such as news articles, personal pronouns
8169 can be resolved only by determining the speaker for each mention (Lee et al., 2013).
8170 Coreference is also more likely between mentions from the same speaker.

8171 Gazetteers. These features indicate that the anaphor and candidate antecedent appear in a
8172 gazetteer of acronyms (e.g., USA/United States, GATech/Georgia Tech), demonymns
8173 (e.g., Israel/Israeli), or other aliases (e.g., Knickerbockers/New York Knicks).

8174 Lexical semantics. These features use a lexical resource such as WordNet to determine
8175 whether the head words of the mentions are related through synonymy, antonymy,
8176 and hypernymy (§ 4.2).

8177 Dependency paths. The dependency path between the anaphor and candidate antecedent
8178 can help to determine whether the pair can corefer, under the government and binding
8179 constraints described in § 15.1.1.

8180 Comprehensive lists of mention-pair features are offered by Bengtson and Roth (2008) and
8181 Rahman and Ng (2011). Neural network approaches use far fewer mention-pair features:
8182 for example, Lee et al. (2017) include only speaker, genre, distance, and mention width
8183 features.

8184 Semantics In many cases, coreference seems to require knowledge and semantic inferences,
8185 as in the running example, where we link China with a country and a growth market. Some
8186 of this information can be gleaned from WordNet, which defines a graph over synsets (see
8187 § 4.2). For example, one of the synsets of China is an instance of an Asian_nation#1, which
8188 in turn is a hyponym of country#2, a synset that includes country.⁷ Such paths can be
8189 used to measure the similarity between concepts (Pedersen et al., 2004), and this similarity
8190 can be incorporated into coreference resolution as a feature (Ponzetto and Strube, 2006).

⁷teletype font is used to indicate wordnet synsets, and italics is used to indicate strings.

8191 Similar ideas can be applied to knowledge graphs induced from Wikipedia (Ponzetto and
 8192 Strube, 2007). But while such approaches improve relatively simple classification-based
 8193 systems, they have proven less useful when added to the current generation of techniques.⁸
 8194 For example, Durrett and Klein (2013) employ a range of semantics-based features —
 8195 WordNet synonymy and hypernymy relations on head words, named entity types (e.g.,
 8196 person, organization), and unsupervised clustering over nominal heads — but find that
 8197 these features give minimal improvement over a baseline system using surface features.

8198 15.3.1.3 Entity features

8199 Many of the features for entity-mention coreference are generated by aggregating mention-pair
 8200 features over all mentions in the candidate entity (Culotta et al., 2007; Rahman and Ng,
 8201 2011). Specifically, for each binary mention-pair feature $f(i, j)$, we compute the following
 8202 entity-mention features for mention i and entity $e = \{j : j < i \wedge z_j = e\}$.

- 8203 • All-True: Feature $f(i, j)$ holds for all mentions $j \in e$.
- 8204 • Most-True: Feature $f(i, j)$ holds for at least half and fewer than all mentions $j \in e$.
- 8205 • Most-False: Feature $f(i, j)$ holds for at least one and fewer than half of all mentions
 8206 $j \in e$.
- 8207 • None: Feature $f(i, j)$ does not hold for any mention $j \in e$.

8208 For scalar mention-pair features (e.g., distance features), aggregation can be performed by
 8209 computing the minimum, maximum, and median values across all mentions in the cluster.
 8210 Additional entity-mention features include the number of mentions currently clustered in
 8211 the entity, and All-X and Most-X features for each mention type.

8212 15.3.2 Distributed representations of mentions and entities

8213 Recent work has emphasized distributed representations of both mentions and entities.
 8214 One potential advantage is that pre-trained embeddings could help to capture the semantic
 8215 compatibility underlying nominal coreference, helping with difficult cases like (Apple, the firm)
 8216 and (China, the firm’s biggest growth market). Furthermore, a distributed representation
 8217 of entities can be trained to capture semantic features that are added by each mention.

8218 15.3.2.1 Mention embeddings

8219 Entity mentions can be embedded into a vector space, providing the base layer for neural
 8220 networks that score coreference decisions (Wiseman et al., 2015).

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.

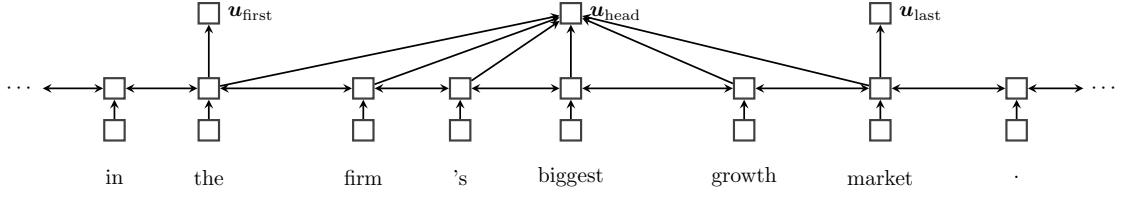


Figure 15.5: A bidirectional recurrent model of mention embeddings. The mention is represented by its first word, its last word, and an estimate of its head word, which is computed from a weighted average (Lee et al., 2017).

8221 Constructing the mention embedding Various approaches for embedding multiword units
 8222 can be applied (see § 14.8). Figure 15.5 shows a recurrent neural network approach, which
 8223 begins by running a bidirectional LSTM over the entire text, obtaining hidden states from
 8224 the left-to-right and right-to-left passes, $\mathbf{h}_m = [\overleftarrow{\mathbf{h}}_m; \overrightarrow{\mathbf{h}}_m]$. Each candidate mention span
 8225 (s, t) is then represented by the vertical concatenation of four vectors:

$$\mathbf{u}^{(s,t)} = [\mathbf{u}_{\text{first}}^{(s,t)}; \mathbf{u}_{\text{last}}^{(s,t)}; \mathbf{u}_{\text{head}}^{(s,t)}; \phi^{(s,t)}], \quad [15.26]$$

8226 where $\mathbf{u}_{\text{first}}^{(s,t)} = \mathbf{h}_{s+1}$ is the embedding of the first word in the span, $\mathbf{u}_{\text{last}}^{(s,t)} = \mathbf{h}_t$ is the
 8227 embedding of the last word, $\mathbf{u}_{\text{head}}^{(s,t)}$ is the embedding of the “head” word, and $\phi^{(s,t)}$ is a
 8228 vector of surface features, such as the length of the span (Lee et al., 2017).

Attention over head words Rather than identifying the head word from the output of a parser, it can be computed from a neural attention mechanism:

$$\tilde{\alpha}_m = \theta_\alpha \cdot \mathbf{h}_m \quad [15.27]$$

$$\mathbf{a}^{(s,t)} = \text{SoftMax}([\tilde{\alpha}_{s+1}, \tilde{\alpha}_{s+2}, \dots, \tilde{\alpha}_t]) \quad [15.28]$$

$$\mathbf{u}_{\text{head}}^{(s,t)} = \sum_{m=s+1}^t a_m^{(s,t)} \mathbf{h}_m. \quad [15.29]$$

8229 Each token m gets a scalar score $\tilde{\alpha}_m = \theta_\alpha \cdot \mathbf{h}_m$, which is the dot product of the LSTM
 8230 hidden state \mathbf{h}_m and a vector of weights θ_α . The vector of scores for tokens in the
 8231 span $m \in \{s + 1, s + 2, \dots, t\}$ is then passed through a softmax layer, yielding a vector
 8232 $\mathbf{a}^{(s,t)}$ that allocates one unit of attention across the span. This eliminates the need for
 8233 syntactic parsing to recover the head word; instead, the model learns to identify the most
 8234 important words in each span. Attention mechanisms were introduced in neural machine
 8235 translation (Bahdanau et al., 2014), and are described in more detail in § 18.3.1.

Using mention embeddings Given a set of mention embeddings, each mention i and candidate antecedent a is scored as,

$$\psi(a, i) = \psi_S(a) + \psi_S(i) + \psi_M(a, i) \quad [15.30]$$

$$\psi_S(a) = \text{FeedForward}_S(\mathbf{u}^{(a)}) \quad [15.31]$$

$$\psi_S(i) = \text{FeedForward}_S(\mathbf{u}^{(i)}) \quad [15.32]$$

$$\psi_M(a, i) = \text{FeedForward}_M([\mathbf{u}^{(a)}; \mathbf{u}^{(i)}; \mathbf{u}^{(a)} \odot \mathbf{u}^{(i)}; \mathbf{f}(a, i, \mathbf{w})]), \quad [15.33]$$

where $\mathbf{u}^{(a)}$ and $\mathbf{u}^{(i)}$ are the embeddings for spans a and i respectively, as defined in Equation 15.26.

- The scores $\psi_S(a)$ quantify whether span a is likely to be a coreferring mention, independent of what it corefers with. This allows the model to learn identify mentions directly, rather than identifying mentions with a preprocessing step.
- The score $\psi_M(a, i)$ computes the compatibility of spans a and i . Its base layer is a vector that includes the embeddings of spans a and i , their elementwise product $\mathbf{u}^{(a)} \odot \mathbf{u}^{(i)}$, and a vector of surface features $\mathbf{f}(a, i, \mathbf{w})$, including distance, speaker, and genre information.

Lee et al. (2017) provide an error analysis that shows how this method can correctly link a blaze and a fire, while incorrectly linking pilots and fight attendants. In each case, the coreference decision is based on similarities in the word embeddings.

Rather than embedding individual mentions, Clark and Manning (2016) embed mention pairs. At the base layer, their network takes embeddings of the words in and around each mention, as well as one-hot vectors representing a few surface features, such as the distance and string matching features. This base layer is then passed through a multilayer feedforward network with ReLU nonlinearities, resulting in a representation of the mention pair. The output of the mention pair encoder $\mathbf{u}_{i,j}$ is used in the scoring function of a mention-ranking model, $\psi_M(i, j) = \boldsymbol{\theta} \cdot \mathbf{u}_{i,j}$. 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

8264 is updated recurrently, $\mathbf{v}_e \leftarrow f(\mathbf{v}_e, \mathbf{u}_i)$, using a recurrent neural network such as a long
8265 short-term memory (LSTM; chapter 6). Alternatively, we can apply a pooling operation,
8266 such as max-pooling or average-pooling (chapter 3), setting $\mathbf{v}_e \leftarrow \text{Pool}(\mathbf{v}_e, \mathbf{u}_i)$. In either
8267 case, the update to the representation of entity e can be thought of as adding new
8268 information about the entity from mention i .

8269

15.4 Evaluating coreference resolution

8270 The state of coreference evaluation is aggravatingly complex. Early attempts at simple
8271 evaluation metrics were found to under-penalize trivial baselines, such as placing each
8272 mention in its own cluster, or grouping all mentions into a single cluster. Following Denis
8273 and Baldwin (2009), the CoNLL 2011 shared task on coreference (Pradhan et al., 2011)
8274 formalized the practice of averaging across three different metrics: Muc (Vilain et al., 1995),
8275 B-Cubed (Bagga and Baldwin, 1998a), and Ceaf (Luo, 2005). Reference implementations
8276 of these metrics are available from Pradhan et al. (2014) at <https://github.com/conll/reference-coreference-scorers>.

8278

Additional resources

8279 Ng (2010) surveys coreference resolution through 2010. Early work focused exclusively on
8280 pronoun resolution, with rule-based (Lappin and Leass, 1994) and probabilistic methods (Ge
8281 et al., 1998). The full coreference resolution problem was popularized in a shared task
8282 associated with the sixth Message Understanding Conference, which included coreference
8283 annotations for training and test sets of thirty documents each (Grishman and Sundheim,
8284 1996). An influential early paper was the decision tree approach of Soon et al. (2001),
8285 who introduced mention ranking. A comprehensive list of surface features for coreference
8286 resolution is offered by Bengtson and Roth (2008). Durrett and Klein (2013) improved on
8287 prior work by introducing a large lexicalized feature set; subsequent work has emphasized
8288 neural representations of entities and mentions (Wiseman et al., 2015).

8289

Exercises

- 8290 1. Select an article from today’s news, and annotate coreference for the first twenty noun
8291 phrases that appear in the article (include nested noun phrases). That is, group the
8292 noun phrases into entities, where each entity corresponds to a set of noun phrases.
8293 Then specify the mention-pair training data that would result from the first five noun
8294 phrases.
- 8295 2. Using your annotations from the preceding problem, compute the following statistics:

- 8296 • The number of times new entities are introduced by each of the three types of
 8297 referring expressions: pronouns, proper nouns, and nominals. Include “singleton”
 8298 entities that are mentioned only once.
 - 8299 • For each type of referring expression, compute the fraction of mentions that are
 8300 anaphoric.
- 8301 3. Apply a simple heuristic to all pronouns in the article from the previous exercise.
 8302 Specifically, link each pronoun to the closest preceding noun phrase that agrees in
 8303 gender, number, animacy, and person. Compute the following evaluation:
- 8304 • True positive: a pronoun that is linked to a noun phrase with which it is
 8305 coreferent, or is correctly labeled as the first mention of an entity.
 - 8306 • False positive: a pronoun that is linked to a noun phrase with which it is
 8307 not coreferent. (This includes mistakenly linking singleton or non-referential
 8308 pronouns.)
 - 8309 • False negative: a pronoun that is not linked to a noun phrase with which it is
 8310 coreferent.
- 8311 Compute the F -measure for your method, and for a trivial baseline in which every
 8312 mention is its own entity. Are there any additional heuristics that would have
 8313 improved the performance of this method?
- 8314 4. Durrett and Klein (2013) compute the probability of the gold coreference clustering
 8315 by summing over all antecedent structures that are compatible with the clustering.
 8316 Compute the number of antecedent structures for a single entity with K mentions.
- 8317 5. Use the policy gradient algorithm to compute the gradient for the following scenario,
 8318 based on the Bell tree in Figure 15.4:
- 8319 • The gold clustering \mathbf{c}^* is $\{\text{Abigail}, \text{her}\}, \{\text{she}\}$.
 - 8319 • Drawing a single sequence of actions ($K = 1$) from the current policy, you obtain
 8319 the following incremental clusterings:

$$\begin{aligned} \mathbf{c}(a_1) &= \{\text{Abigail}\} \\ \mathbf{c}(\mathbf{a}_{1:2}) &= \{\text{Abigail}, \text{she}\} \\ \mathbf{c}(\mathbf{a}_{1:3}) &= \{\text{Abigail}, \text{she}\}, \{\text{her}\}. \end{aligned}$$

- 8320 • At each mention t , the action space \mathcal{A}_t is to merge the mention with each existing
 8321 cluster, or the empty cluster, with probability,

$$\Pr(\text{Merge}(m_t, \mathbf{c}(\mathbf{a}_{1:t-1}))) \propto \exp \psi_E(m_t \cup \mathbf{c}(\mathbf{a}_{1:t-1})), \quad [15.34]$$

8322 where the cluster score $\psi_E(m_t \cup c)$ is defined in Equation 15.15.

8323 Compute the gradient $\frac{\partial}{\partial \theta} L(\theta)$ in terms of the loss $\ell(c(a))$ and the features of each
8324 (potential) cluster. Explain the differences between the gradient-based update $\theta \leftarrow \theta - \frac{\partial}{\partial \theta} L(\theta)$
8325 and the incremental perceptron update from this sample example.

8326 Chapter 16

8327 Discourse

8328 Applications of natural language processing often concern multi-sentence documents: from
8329 paragraph-long restaurant reviews, to 500-word newspaper articles, to 500-page novels.
8330 Yet most of the methods that we have discussed thus far are concerned with individual
8331 sentences. This chapter discusses theories and methods for handling multi-sentence linguistic
8332 phenomena, known collectively as discourse. There are diverse characterizations of discourse
8333 structure, and no single structure is ideal for every computational application. This
8334 chapter covers some of the most well studied discourse representations, while highlighting
8335 computational models for identifying and exploiting these structures.

8336 16.1 Segments

8337 A document or conversation can be viewed as a sequence of segments, each of which is
8338 cohesive in its content and/or function. In Wikipedia biographies, these segments often
8339 pertain to various aspects to the subject's life: early years, major events, impact on others,
8340 and so on. This segmentation is organized around topics. Alternatively, scientific research
8341 articles are often organized by functional themes: the introduction, a survey of previous
8342 research, experimental setup, and results.

8343 Written texts often mark segments with section headers and related formatting devices.
8344 However, such formatting may be too coarse-grained to support applications such as
8345 the retrieval of specific passages of text that are relevant to a query (Hearst, 1997).
8346 Unformatted speech transcripts, such as meetings and lectures, are also an application
8347 scenario for segmentation (Carletta, 2007; Glass et al., 2007; Janin et al., 2003).

8348 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

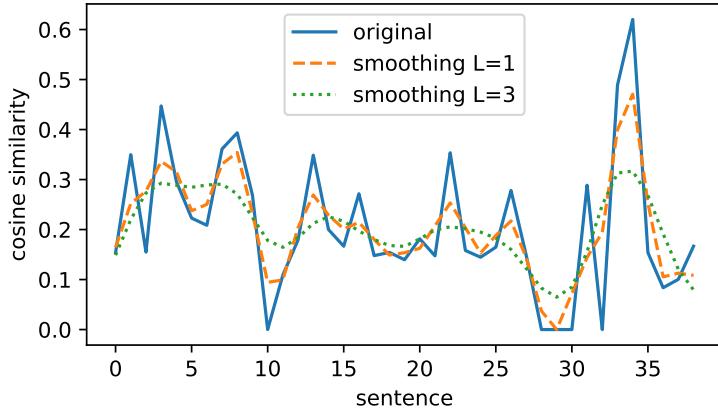


Figure 16.1: Smoothed cosine similarity among adjacent sentences in a news article. Local minima at $m = 10$ and $m = 29$ indicate likely segmentation points.

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,

$$s_m = \frac{\mathbf{x}_m \cdot \mathbf{x}_{m+1}}{\|\mathbf{x}_m\|_2 \times \|\mathbf{x}_{m+1}\|_2} \quad [16.1]$$

$$\bar{s}_m = \sum_{\ell=0}^L k_\ell (s_{m+\ell} + s_{m-\ell}), \quad [16.2]$$

with k_ℓ representing the value of a smoothing kernel of size L , e.g. $\mathbf{k} = [1, 0.5, 0.25]^\top$. Segmentation points are then identified at local minima in the smoothed similarities \bar{s} , since these points indicate changes in the overall distribution of words in the text. An example is shown in Figure 16.1.

Text segmentation can also be formulated as a probabilistic model, in which each segment has a unique language model that defines the probability over the text in the segment (Utiyama and Isahara, 2001; Eisenstein and Barzilay, 2008; Du et al., 2013).¹ A good segmentation achieves high likelihood by grouping segments with similar word distributions. This probabilistic approach can be extended to hierarchical topic segmentation,

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.

8358 in which each topic segment is divided into subsegments (Eisenstein, 2009). All of these
 8359 approaches are unsupervised. While labeled data can be obtained from well-formatted texts
 8360 such as textbooks, such annotations may not generalize to speech transcripts in alternative
 8361 domains. Supervised methods have been tried in cases where in-domain labeled data is
 8362 available, substantially improving performance by learning weights on multiple types of
 8363 features (Galley et al., 2003).

8364 16.1.2 Functional segmentation

8365 In some genres, there is a canonical set of communicative functions: for example, in
 8366 scientific research articles, one such function is to communicate the general background
 8367 for the article, another is to introduce a new contribution, or to describe the aim of
 8368 the research (Teufel et al., 1999). A functional segmentation divides the document into
 8369 contiguous segments, sometimes called rhetorical zones, in which each sentence has the
 8370 same function. Teufel and Moens (2002) train a supervised classifier to identify the
 8371 functional of each sentence in a set of scientific research articles, using features that
 8372 describe the sentence’s position in the text, its similarity to the rest of the article and
 8373 title, tense and voice of the main verb, and the functional role of the previous sentence.
 8374 Functional segmentation can also be performed without supervision. Noting that some
 8375 types of Wikipedia articles have very consistent functional segmentations (e.g., articles
 8376 about cities or chemical elements), Chen et al. (2009) introduce an unsupervised model
 8377 for functional segmentation, which learns both the language model associated with each
 8378 function and the typical patterning of functional segments across the article.

8379 16.2 Entities and reference

8380 Another dimension of discourse relates to which entities are mentioned throughout the text,
 8381 and how. Consider the examples in Figure 16.2: Grosz et al. (1995) argue that the first

discourse is more coherent. Do you agree? The examples differ in their choice of referring expressions for the protagonist John, and in the syntactic constructions in sentences (b) and (d). The examples demonstrate the need for theoretical models to explain how referring expressions are chosen, and where they are placed within sentences. Such models can then be used to help interpret the overall structure of the discourse, to measure discourse coherence, and to generate discourses in which referring expressions are used coherently.

16.2.1 Centering theory

The relationship between discourse and entity reference is most elaborated in centering theory (Grosz et al., 1995). According to the theory, every utterance in the discourse is characterized by a set of entities, known as centers.

- The forward-looking centers in utterance m are all the entities that are mentioned in the utterance, $c_f(\mathbf{w}_m) = \{e_1, e_2, \dots\}$. The forward-looking centers are partially ordered by their syntactic prominence, favoring subjects over other positions.
- The backward-looking center $c_b(\mathbf{w}_m)$ is the highest-ranked element in the set of forward-looking centers from the previous utterance $c_f(\mathbf{w}_{m-1})$ that is also mentioned in \mathbf{w}_m .

Given these two definitions, centering theory makes the following predictions about the form and position of referring expressions:

1. If a pronoun appears in the utterance \mathbf{w}_m , then the backward-looking center $c_b(\mathbf{w}_m)$ must also be realized as a pronoun. This rule argues against the use of it to refer to the piano store in Example (16.2d), since John is the backward looking center of (16.2d), and he is mentioned by name and not by a pronoun.
2. Sequences of utterances should retain the same backward-looking center if possible, and ideally, the backward-looking center should also be the top-ranked element in the list of forward-looking centers. This rule argues in favor of the preservation of John as the backward-looking center throughout Example (16.1).

Centering theory unifies aspects of syntax, discourse, and anaphora resolution. However, it can be difficult to clarify exactly how to rank the elements of each utterance, or even how to partition a text or dialog into utterances (Poesio et al., 2004).

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

	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.

8414 column per entity (Barzilay and Lapata, 2008). Each cell $c(m, i)$ can take the following
 8415 values:

$$c(m, i) = \begin{cases} S, & \text{entity } i \text{ is in subject position in sentence } m \\ O, & \text{entity } i \text{ is in object position in sentence } m \\ X, & \text{entity } i \text{ appears in sentence } m, \text{ in neither subject nor object position} \\ -, & \text{entity } i \text{ does not appear in sentence } m. \end{cases} \quad [16.3]$$

8416 To populate the entity grid, syntactic parsing is applied to identify subject and object
 8417 positions, and coreference resolution is applied to link multiple mentions of a single entity.
 8418 An example is shown in Figure 16.3.

8419 After the grid is constructed, the coherence of a document can be measured by the
 8420 transitions between adjacent cells in each column. For example, the transition $(S \rightarrow S)$
 8421 keeps an entity in subject position across adjacent sentences; the transition $(O \rightarrow S)$
 8422 promotes an entity from object position to subject position; the transition $(S \rightarrow -)$ drops
 8423 the subject of one sentence from the next sentence. The probabilities of each transition
 8424 can be estimated from labeled data, and an entity grid can then be scored by the sum
 8425 of the log-probabilities across all columns and all transitions, $\sum_{i=1}^{N_e} \sum_{m=1}^M \log p(c(m, i) |$
 8426 $c(m-1, i))$. The resulting probability can be used as a proxy for the coherence of a text.
 8427 This has been shown to be useful for a range of tasks: determining which of a pair of
 8428 articles is more readable (Schwartz and Ostendorf, 2005), correctly ordering the sentences
 8429 in a scrambled text (Lapata, 2003), and disentangling multiple conversational threads in
 8430 an online multi-party chat (Elsner and Charniak, 2010).

8431 16.2.3 *Formal semantics beyond the sentence level

8432 An alternative view of the role of entities in discourse focuses on formal semantics, and the
 8433 construction of meaning representations for multi-sentence units. Consider the following
 8434 two sentences (from Bird et al., 2009):

- 8435 (16.3) a. Angus owns a dog.
 8436 b. It bit Irene.

8437 We would like to recover the formal semantic representation,

$$\exists x.\text{dog}(x) \wedge \text{own}(\text{Angus}, x) \wedge \text{bite}(x, \text{Irene}). \quad [16.4]$$

However, the semantic representations of each individual sentence are:

$$\exists x.\text{dog}(x) \wedge \text{own}(\text{Angus}, x) \quad [16.5]$$

$$\text{bite}(y, \text{Irene}). \quad [16.6]$$

8438 Unifying these two representations into the form of Equation 16.4 requires linking the
 8439 unbound variable y from [16.6] with the quantified variable x in [16.5]. Discourse understanding
 8440 therefore requires the reader to update a set of assignments, from variables to entities. This
 8441 update would (presumably) link the dog in the first sentence of [16.3] with the unbound
 8442 variable y in the second sentence, thereby licensing the conjunction in [16.4].² This basic
 8443 idea is at the root of dynamic semantics (Groenendijk and Stokhof, 1991). Segmented
 8444 discourse representation theory links dynamic semantics with a set of discourse relations,
 8445 which explain how adjacent units of text are rhetorically or conceptually related (Lascarides
 8446 and Asher, 2007). The next section explores the theory of discourse relations in more detail.

8447 16.3 Relations

8448 In dependency grammar, sentences are characterized by a graph (usually a tree) of syntactic
 8449 relations between words, such as Nsubj and Det. A similar idea can be applied at the
 8450 document level, identifying relations between discourse units, such as clauses, sentences,
 8451 or paragraphs. The task of discourse parsing involves identifying discourse units and the
 8452 relations that hold between them. These relations can then be applied to tasks such as
 8453 document classification and summarization, as discussed in § 16.3.4.

8454 16.3.1 Shallow discourse relations

8455 The existence of discourse relations is hinted by discourse connectives, such as however,
 8456 moreover, meanwhile, and if ...then. These connectives explicitly specify the relationship

²This linking task is similar to coreference resolution (see chapter 15), but here the connections are between semantic variables, rather than spans of text.

- TEMPORAL
 - Asynchronous
 - Synchronous:
precedence, succession
- CONTINGENCY
 - Cause: result, reason
 - Pragmatic cause:
justification
 - Condition: hypothetical,
general, unreal present,
unreal past, real present,
real past
 - Pragmatic condition:
relevance, implicit
assertion
- COMPARISON
 - Contrast: juxtaposition, opposition
 - Pragmatic contrast
 - Concession: expectation,
contra-expectation
 - Pragmatic concession
- EXPANSION
 - Conjunction
 - Instantiation
 - Restatement: specification,
equivalence, generalization
 - Alternative: conjunctive, disjunctive,
chosen alternative
 - Exception
 - List

Table 16.1: The hierarchy of discourse relation in the Penn Discourse Treebank annotations (Prasad et al., 2008). For example, precedence is a subtype of Synchronous, which is a type of temporal relation.

8457 between adjacent units of text: however signals a contrastive relationship, moreover signals
 8458 that the subsequent text elaborates or strengthens the point that was made immediately
 8459 beforehand, meanwhile indicates that two events are contemporaneous, and if ...then sets
 8460 up a conditional relationship. Discourse connectives can therefore be viewed as a starting
 8461 point for the analysis of discourse relations.

8462 In lexicalized tree-adjoining grammar for discourse (D-LTAG), each connective anchors
 8463 a relationship between two units of text (Webber, 2004). This model provides the theoretical
 8464 basis for the Penn Discourse Treebank (PDTB), the largest corpus of discourse relations in
 8465 English (Prasad et al., 2008). It includes a hierarchical inventory of discourse relations
 8466 (shown in Table 16.1), which is created by abstracting the meanings implied by the
 8467 discourse connectives that appear in real texts (Knott, 1996). These relations are then
 8468 annotated on the same corpus of news text used in the Penn Treebank (see § 9.2.2), adding
 8469 the following information:

- 8470 • Each connective is annotated for the discourse relation or relations that it expresses, if
 8471 any — many discourse connectives have senses in which they do not signal a discourse
 8472 relation (Pitler and Nenkova, 2009).

- (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).

- For each discourse relation, the two arguments of the relation are specified as arg1 and arg2, where arg2 is constrained to be adjacent to the connective. These arguments may be sentences, but they may also smaller or larger units of text.
 - Adjacent sentences are annotated for implicit discourse relations, which are not marked by any connective. When a connective could be inserted between a pair of sentence, the annotator supplies it, and also labels its sense (e.g., example 16.5). In some cases, there is no relationship at all between a pair of adjacent sentences; in other cases, the only relation is that the adjacent sentences mention one or more shared entity. These phenomena are annotated as NoRel and EntRel (entity relation), respectively.
- Examples of Penn Discourse Treebank annotations are shown in (16.4). In (16.4), the word therefore acts as an explicit discourse connective, linking the two adjacent units of text. The Treebank annotations also specify the “sense” of each relation, linking the connective to a relation in the sense inventory shown in Table 16.1: in (16.4), the relation is pragmatic cause:justification because it relates to the author’s communicative intentions. The word therefore can also signal causes in the external world (e.g., He was therefore forced to relinquish his plan). In discourse sense classification, the goal is to determine which discourse relation, if any, is expressed by each connective. A related task is the classification of implicit discourse relations, as in (16.5). In this example, the relationship between the adjacent sentences could be expressed by the connective because, indicating a cause:reason relationship.

8494 16.3.1.1 Classifying explicit discourse relations and their arguments

8495 As suggested by the examples above, many connectives can be used to invoke multiple
 8496 types of discourse relations. Similarly, some connectives have senses that are unrelated to
 8497 discourse: for example, and functions as a discourse connective when it links propositions,
 8498 but not when it links noun phrases (Lin et al., 2014). Nonetheless, the senses of explicitly-marked
 8499 discourse relations in the Penn Treebank are relatively easy to classify, at least at the
 8500 coarse-grained level. When classifying the four top-level PDTB relations, 90% accuracy
 8501 can be obtained simply by selecting the most common relation for each connective (Pitler
 8502 and Nenkova, 2009). At the more fine-grained levels of the discourse relation hierarchy,
 8503 connectives are more ambiguous. This fact is reflected both in the accuracy of automatic
 8504 sense classification (Versley, 2011) and in interannotator agreement, which falls to 80% for
 8505 level-3 discourse relations (Prasad et al., 2008).

8506 A more challenging task for explicitly-marked discourse relations is to identify the
 8507 scope of the arguments. Discourse connectives need not be adjacent to arg1, as shown in
 8508 item 16.6, where arg1 follows arg2; furthermore, the arguments need not be contiguous, as
 8509 shown in (16.7). For these reasons, recovering the arguments of each discourse connective is
 8510 a challenging subtask. Because intra-sentential arguments are often syntactic constituents
 8511 (see chapter 10), many approaches train a classifier to predict whether each constituent is
 8512 an appropriate argument for each explicit discourse connective (Wellner and Pustejovsky,
 8513 2007; Lin et al., 2014, e.g.,).

8514 16.3.1.2 Classifying implicit discourse relations

Implicit discourse relations are considerably more difficult to classify and to annotate.³
 Most approaches are based on an encoding of each argument, which is then used as input
 to a non-linear classifier:

$$\mathbf{z}^{(i)} = \text{Encode}(\mathbf{w}^{(i)}) \quad [16.7]$$

$$\mathbf{z}^{(i+1)} = \text{Encode}(\mathbf{w}^{(i+1)}) \quad [16.8]$$

$$\hat{y}_i = \underset{y}{\operatorname{argmax}} \Psi(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}). \quad [16.9]$$

8515 This basic framework can be instantiated in several ways, including both feature-based
 8516 and neural encoders. Several recent approaches are compared in the 2015 and 2016 shared
 8517 tasks at the Conference on Natural Language Learning (Xue et al., 2015, 2016).

8518 Feature-based approaches Each argument can be encoded into a vector of surface features.
 8519 The encoding typically includes lexical features (all words, or all content words, or a subset

³In the dataset for the 2015 shared task on shallow discourse parsing, the interannotator agreement was 91% for explicit discourse relations and 81% for non-explicit relations, across all levels of detail (Xue et al., 2015).

of words such as the first three and the main verb), Brown clusters of individual words (§ 14.4), and syntactic features such as terminal productions and dependency arcs (Pitler et al., 2009; Lin et al., 2009; Rutherford and Xue, 2014). The classification function then has two parts. First, it creates a joint feature vector by combining the encodings of each argument, typically by computing the cross-product of all features in each encoding:

$$\mathbf{f}(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}) = \{(a \times b \times y) : (\mathbf{z}_a^{(i)} \mathbf{z}_b^{(i+1)})\} \quad [16.10]$$

The size of this feature set grows with the square of the size of the vocabulary, so it can be helpful to select a subset of features that are especially useful on the training data (Park and Cardie, 2012). After \mathbf{f} is computed, any classifier can be trained to compute the final score, $\Psi(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}) = \boldsymbol{\theta} \cdot \mathbf{f}(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)})$.

Neural network approaches In neural network architectures, the encoder is learned jointly with the classifier as an end-to-end model. Each argument can be encoded using a variety of neural architectures (surveyed in § 14.8): recursive (§ 10.6.1; Ji and Eisenstein, 2015), recurrent (§ 6.3; Ji et al., 2016), and convolutional (§ 3.4; Qin et al., 2017). The classification function can then be implemented as a feedforward neural network on the two encodings (chapter 3; for examples, see Rutherford et al., 2017; Qin et al., 2017), or as a simple bilinear product, $\Psi(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}) = (\mathbf{z}^{(i)})^\top \boldsymbol{\Theta}_y \mathbf{z}^{(i+1)}$ (Ji and Eisenstein, 2015). The encoding model can be trained by backpropagation from the classification objective, such as the margin loss. Rutherford et al. (2017) show that neural architectures outperform feature-based approaches in most settings. While neural approaches require engineering the network architecture (e.g., embedding size, number of hidden units in the classifier), feature-based approaches also require significant engineering to incorporate linguistic resources such as Brown clusters and parse trees, and to select a subset of relevant features.

16.3.2 Hierarchical discourse relations

In sentence parsing, adjacent phrases combine into larger constituents, ultimately producing a single constituent for the entire sentence. The resulting tree structure enables structured analysis of the sentence, with subtrees that represent syntactically coherent chunks of meaning. Rhetorical Structure Theory (RST) extends this style of hierarchical analysis to the discourse level (Mann and Thompson, 1988).

The basic element of RST is the discourse unit, which refers to a contiguous span of text. Elementary discourse units (EDUs) are the atomic elements in this framework, and are typically (but not always) clauses.⁴ Each discourse relation combines two or more

⁴Details of discourse segmentation can be found in the RST annotation manual (Carlson and Marcu, 2001).

8552 adjacent discourse units into a larger, composite discourse unit; this process ultimately
 8553 unites the entire text into a tree-like structure.⁵

8554 Nuclearity In many discourse relations, one argument is primary. For example:

8555 (16.8) [LaShawn loves animals]_N
 8556 [She has nine dogs and one pig]_S

8557 In this example, the second sentence provides evidence for the point made in the first
 8558 sentence. The first sentence is thus the nucleus of the discourse relation, and the second
 8559 sentence is the satellite. The notion of nuclearity is analogous to the head-modifier structure
 8560 of dependency parsing (see § 11.1.1). However, in RST, some relations have multiple nuclei.
 8561 For example, the arguments of the contrast relation are equally important:

8562 (16.9) [The clash of ideologies survives this treatment]_N
 8563 [but the nuance and richness of Gorky's individual characters have vanished in the scuffle]_N⁶

8564 Relations that have multiple nuclei are called coordinating; relations with a single nucleus
 8565 are called subordinating. Subordinating relations are constrained to have only two arguments,
 8566 while coordinating relations (such as conjunction) may have more than two.

8567 RST Relations Rhetorical structure theory features a large inventory of discourse relations,
 8568 which are divided into two high-level groups: subject matter relations, and presentational
 8569 relations. Presentational relations are organized around the intended beliefs of the reader.
 8570 For example, in (16.8), the second discourse unit provides evidence intended to increase
 8571 the reader's belief in the proposition expressed by the first discourse unit, that LaShawn
 8572 loves animals. In contrast, subject-matter relations are meant to communicate additional
 8573 facts about the propositions contained in the discourse units that they relate:

8574 (16.10) [the debt plan was rushed to completion]_N
 8575 [in order to be announced at the meeting]_S⁷

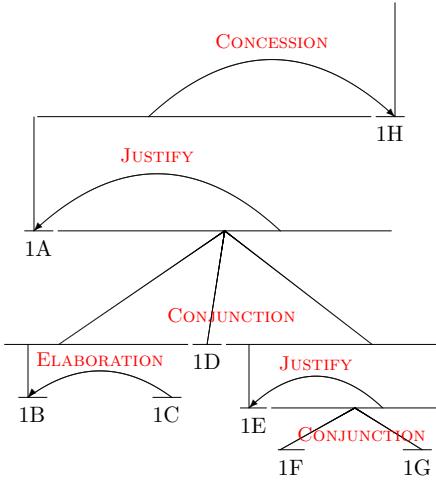
⁵While RST analyses are typically trees, this should be taken as a strong theoretical commitment to the principle that all coherent discourses have a tree structure. Taboada and Mann (2006) write:

It is simply the case that trees are convenient, easy to represent, and easy to understand.
 There is, on the other hand, no theoretical reason to assume that trees are the only possible representation of discourse structure and of coherence relations.

The appropriateness of tree structures to discourse has been challenged, e.g., by Wolf and Gibson (2005), who propose a more general graph-structured representation.

⁶from the RST Treebank (Carlson et al., 2002)

⁷from 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.

8576 In this example, the satellite describes a world state that is realized by the action described
 8577 in the nucleus. This relationship is about the world, and not about the author’s communicative
 8578 intentions.

8579 Example Figure 16.5 depicts an RST analysis of a paragraph from a movie review.
 8580 Asymmetric (subordinating) relations are depicted with an arrow from the satellite to the nucleus; symmetric (coordinating) relations are depicted with lines. The elementary
 8581 discourse units 1F and 1G are combined into a larger discourse unit with the symmetric
 8582 Conjunction relation. The resulting discourse unit is then the satellite in a Justify relation
 8583 with 1E.
 8584

8585 16.3.2.1 Hierarchical discourse parsing

8586 The goal of discourse parsing is to recover a hierarchical structural analysis from a document
 8587 text, such as the analysis in Figure 16.5. For now, let’s assume a segmentation of the
 8588 document into elementary discourse units (EDUs); segmentation algorithms are discussed
 8589 below. After segmentation, discourse parsing can be viewed as a combination of two
 8590 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.⁸ Some discourse parsers rely on the strong compositionality criterion (Marcu, 1996), which states the assumption 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 $\{\mathbf{z}^{(i)}\}_{i=1}^N$, and assume a composition function $\text{Compose}(\mathbf{z}^{(i)}, \mathbf{z}^{(j)}, \ell)$, 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 $\Psi(\mathbf{z}^{(i,k)}, \mathbf{z}^{(k,j)}, \ell)$, 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 $\mathbf{z}^{(i,j,\ell)}$. This representation is then used to combine the discourse unit spanning $i + 1 : j$ in higher-level discourse relations. The representation $\mathbf{z}^{(i,j,\ell)}$ depends on the entire substructure of the unit spanning $i + 1 : j$, and this violates the locality 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 is an

⁸To use these algorithms, is also necessary to binarize all discourse relations during parsing, and then to “unbinarize” them to reconstruct the desired structure (e.g., Hernault et al., 2010).

8626 appealing alternative. The shift-reduce algorithm can be applied to discourse parsing fairly
 8627 directly (Sagae, 2009): the stack stores a set of discourse units and their representations,
 8628 and each action is chosen by a function of these representations. This function could be a
 8629 linear product of weights and features, or it could be a neural network applied to encodings
 8630 of the discourse units. The Reduce action then performs composition on the two discourse
 8631 units at the top of the stack, yielding a larger composite discourse unit, which goes on top
 8632 of the stack. All of the techniques for integrating learning and transition-based parsing,
 8633 described in § 11.3, are applicable to discourse parsing.

8634 16.3.2.2 Segmenting discourse units

8635 In rhetorical structure theory, elementary discourse units do not cross the sentence boundary,
 8636 so discourse segmentation can be performed within sentences, assuming the sentence segmentation
 8637 is given. The segmentation of sentences into elementary discourse units is typically performed
 8638 using features of the syntactic analysis (Braud et al., 2017). One approach is to train a
 8639 classifier to determine whether each syntactic constituent is an EDU, using features such
 8640 as the production, tree structure, and head words (Soricut and Marcu, 2003; Hernault
 8641 et al., 2010). Another approach is to train a sequence labeling model, such as a conditional
 8642 random field (Sporleder and Lapata, 2005; Xuan Bach et al., 2012; Feng et al., 2014). This
 8643 is done using the BIO formalism for segmentation by sequence labeling, described in § 8.3.

8644 16.3.3 Argumentation

8645 An alternative view of text-level relational structure focuses on argumentation (Stab and
 8646 Gurevych, 2014b). Each segment (typically a sentence or clause) may support or rebut
 8647 another segment, creating a graph structure over the text. In the following example (from
 8648 Peldszus and Stede, 2013), segment S_2 provides argumentative support for the proposition
 8649 in the segment S_1 :

- 8650 (16.11) [We should tear the building down,] $_{S_1}$
 8651 [because it is full of asbestos] $_{S_2}$.

8652 Assertions may also support or rebut proposed links between two other assertions, creating
 8653 a hypergraph, which is a generalization of a graph to the case in which edges can join any
 8654 number of vertices. This can be seen by introducing another sentence into the example:

- 8655 (16.12) [In principle it is possible to clean it up,] $_{S_3}$
 8656 [but according to the mayor that is too expensive.] $_{S_4}$

8657 S_3 acknowledges the validity of S_2 , but undercuts its support of S_1 . This can be represented
 8658 by introducing a hyperedge, $(S_3, S_2, S_1)_{\text{undercut}}$, indicating that S_3 undercuts the proposed
 8659 relationship between S_2 and S_1 . S_4 then undercuts the relevance of S_3 .

8660 Argumentation mining is the task of recovering such structures from raw texts. At
8661 present, annotations of argumentation structure are relatively small: Stab and Gurevych
8662 (2014a) have annotated a collection of 90 persuasive essays, and Peldszus and Stede (2015)
8663 have solicited and annotated a set of 112 paragraph-length “microtexts” in German.

8664 16.3.4 Applications of discourse relations

8665 The predominant application of discourse parsing is to select content within a document.
8666 In rhetorical structure theory, the nucleus is considered the more important element of
8667 the relation, and is more likely to be part of a summary of the document; it may also
8668 be more informative for document classification. The D-LTAG theory that underlies the
8669 Penn Discourse Treebank lacks this notion of nuclearity, but arguments may have varying
8670 importance, depending on the relation type. For example, the span of text constituting
8671 arg1 of an expansion relation is more likely to appear in a summary, while the sentence
8672 constituting arg2 of an implicit relation is less likely (Louis et al., 2010). Discourse relations
8673 may also signal segmentation points in the document structure. Explicit discourse markers
8674 have been shown to correlate with changes in subjectivity, and identifying such change
8675 points can improve document-level sentiment classification, by helping the classifier to
8676 focus on the subjective parts of the text (Trivedi and Eisenstein, 2013; Yang and Cardie,
8677 2014).

8678 16.3.4.1 Extractive Summarization

8679 Text summarization is the problem of converting a longer text into a shorter one, while
8680 still conveying the key facts, events, ideas, and sentiments from the original. In extractive
8681 summarization, the summary is a subset of the original text; in abstractive summarization,
8682 the summary is produced de novo, by paraphrasing the original, or by first encoding it into
8683 a semantic representation (see § 19.2). The main strategy for extractive summarization
8684 is to maximize coverage, choosing a subset of the document that best covers the concepts
8685 mentioned in the document as a whole; typically, coverage is approximated by bag-of-words
8686 overlap (Nenkova and McKeown, 2012). Coverage-based objectives can be supplemented
8687 by hierarchical discourse relations, using the principle of nuclearity: in any subordinating
8688 discourse relation, the nucleus is more critical to the overall meaning of the text, and
8689 is therefore more important to include in an extractive summary (Marcu, 1997a).⁹ This
8690 insight can be generalized from individual relations using the concept of discourse depth (Hirao
8691 et al., 2013): for each elementary discourse unit e , the discourse depth d_e is the number of
8692 relations in which a discourse unit containing e is the satellite.

8693 Both discourse depth and nuclearity can be incorporated into extractive summarization,
8694 using constrained optimization. Let \mathbf{x}_n be a bag-of-words vector representation of elementary

⁹Conversely, the arguments of a multi-nuclear relation should either both be included in the summary, or both excluded (Durrett et al., 2016).

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 extractive summarization can be formalized as (Hirao et al., 2013),

$$\begin{aligned} & \max_{\mathbf{y}=\{0,1\}^N} \sum_{n=1}^N y_n \frac{\Psi(\mathbf{x}_n, \{\mathbf{x}_{1:N}\})}{d_n} \\ & \text{s.t. } \sum_{n=1}^N y_n \left(\sum_{j=1}^V x_{n,j} \right) \leq L \\ & \quad y_{\pi(n)} \geq y_n, \quad \forall n \end{aligned} \tag{16.11}$$

where $\Psi(\mathbf{x}_n, \{\mathbf{x}_{1:N}\})$ measures the coverage of elementary discourse unit n with respect 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 constraint ensures that the number of tokens in the summary has an upper bound L . The second constraint ensures that no elementary discourse unit is included unless its parent is also included. In this way, the discourse structure is used twice: to downweight the contributions of elementary discourse units that are not central to the discourse, and to ensure that the resulting structure is a subtree of the original discourse parse. The optimization problem in 16.11 can be solved with integer linear programming, described in § 13.2.2.¹⁰

Figure 16.6 shows a discourse depth tree for the RST analysis from Figure 16.5, in which each elementary discourse is connected to (and below) its parent. The figure also shows a valid summary, corresponding to:

(16.13) It could have been a great movie, and I really liked the son of the leader of the Samurai. But, other than all that, this movie is nothing more than hidden rip-offs.

¹⁰Formally, 16.11 is a special case of the knapsack problem, in which the goal is to find a subset of items with maximum value, constrained by some maximum weight (Cormen et al., 2009).

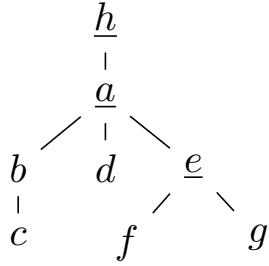


Figure 16.6: A discourse depth tree (Hirao et al., 2013) for the discourse parse from Figure 16.5, in which each elementary discourse unit is connected to its parent. The discourse units in one valid summary are underlined.

8718 16.3.4.2 Document classification

8719 Hierarchical discourse structures lend themselves naturally to text classification: in a
8720 subordinating discourse relation, the nucleus should play a stronger role in the classification
8721 decision than the satellite. Various implementations of this idea have been proposed.

- 8722 • Focusing on within-sentence discourse relations and lexicon-based classification (see
8723 § 4.1.2), Voll and Taboada (2007) simply ignore the text in the satellites of each
8724 discourse relation.
- 8725 • At the document level, elements of each discourse relation argument can be reweighted,
8726 favoring words in the nucleus, and disfavoring words in the satellite (Heerschap et al.,
8727 2011; Bhatia et al., 2015). This approach can be applied recursively, computing
8728 weights across the entire document. The weights can be relation-specific, so that the
8729 features from the satellites of contrastive relations are discounted or even reversed.
- 8730 • Alternatively, the hierarchical discourse structure can define the structure of a recursive
8731 neural network (see § 10.6.1). In this network, the representation of each discourse
8732 unit is computed from its arguments and from a parameter corresponding to the
8733 discourse relation (Ji and Smith, 2017).

8734 Shallow, non-hierarchical discourse relations have also been applied to document classification.
8735 One approach is to impose a set of constraints on the analyses of individual discourse units,
8736 so that adjacent units have the same polarity when they are connected by a discourse
8737 relation indicating agreement, and opposite polarity when connected by a contrastive
8738 discourse relation, indicating disagreement (Somasundaran et al., 2009; Zirn et al., 2011).
8739 Yang and Cardie (2014) apply explicitly-marked relations from the Penn Discourse Treebank
8740 to the problem of sentence-level sentiment polarity classification (see § 4.1). They impose
8741 the following soft constraints:

- When a contrast relation appears between two sentences, those sentences should have opposite sentiment polarity.
- When an expansion or contingency relation appears between two sentences, they should have the same polarity.
- When a contrast relation appears within a sentence, it should have neutral polarity, since it is likely to express both sentiments.

These discourse-driven constraints are shown to improve performance on two datasets of product reviews.

16.3.4.3 Coherence

Just as grammaticality is the property shared by well-structured sentences, coherence is the property shared by well-structured discourses. One application of discourse processing is to measure (and maximize) the coherence of computer-generated texts like translations and summaries (Kibble and Power, 2004). Coherence assessment is also used to evaluate human-generated texts, such as student essays (e.g., Miltzakaki and Kukich, 2004; Burstein et al., 2013).

Coherence subsumes a range of phenomena, many of which have been highlighted earlier in this chapter: e.g., that adjacent sentences should be lexically cohesive (Foltz et al., 1998; Ji et al., 2015; Li and Jurafsky, 2017), and that entity references should follow the principles of centering theory (Barzilay and Lapata, 2008; Nguyen and Joty, 2017). Discourse relations also bear on the coherence of a text in a variety of ways:

- Hierarchical discourse relations tend to have a “canonical ordering” of the nucleus and satellite (Mann and Thompson, 1988): for example, in the elaboration relation from rhetorical structure theory, the nucleus always comes first, while in the justification relation, the satellite tends to be first (Marcu, 1997b).
- Discourse relations should be signaled by connectives that are appropriate to the semantic or functional relationship between the arguments: for example, a coherent text would be more likely to use however to signal a comparison relation than a temporal relation (Kibble and Power, 2004).
- Discourse relations tend to appear in predictable sequences: for example, comparison relations tend to immediately precede contingency relations (Pitler et al., 2008). This observation can be formalized by generalizing the entity grid model (§ 16.2.2), so that each cell (i, j) provides information about the role of the discourse argument containing a mention of entity j in sentence i (Lin et al., 2011). For example, if the first sentence is arg1 of a comparison relation, then any entity mentions in the sentence would be labeled Comp.Arg1. This approach can also be applied to RST discourse relations (Feng et al., 2014).

8778 Datasets One difficulty with evaluating metrics of discourse coherence is that human-generated
8779 texts usually meet some minimal threshold of coherence. For this reason, much of the
8780 research on measuring coherence has focused on synthetic data. A typical setting is to
8781 permute the sentences of a human-written text, and then determine whether the original
8782 sentence ordering scores higher according to the proposed coherence measure (Barzilay
8783 and Lapata, 2008). There are also small datasets of human evaluations of the coherence of
8784 machine summaries: for example, human judgments of the summaries from the participating
8785 systems in the 2003 Document Understanding Conference are available online.¹¹ Researchers
8786 from the Educational Testing Service (an organization which administers several national
8787 exams in the United States) have studied the relationship between discourse coherence
8788 and student essay quality (Burstein et al., 2003, 2010). A public dataset of essays from
8789 second-language learners, with quality annotations, has been made available by researchers
8790 at Cambridge University (Yannakoudakis et al., 2011). At the other extreme, Louis and
8791 Nenkova (2013) analyze the structure of professionally written scientific essays, finding that
8792 discourse relation transitions help to distinguish prize-winning essays from other articles
8793 in the same genre.

8794 Additional resources

8795 For a manuscript-length discussion of discourse processing, see Stede (2011). Article-length
8796 surveys are offered by Webber et al. (2012) and Webber and Joshi (2012).

8797 Exercises

- 8798 1. • Implement the smoothed cosine similarity metric from Equation 16.2, using the
8799 smoothing kernel $\mathbf{k} = [.5, .3, .15, .05]$.
8800 • Download the text of a news article with at least ten paragraphs.
8801 • Compute and plot the smoothed similarity \bar{s} over the length of the article.
8802 • Identify local minima in \bar{s} as follows: first find all sentences m such that $\bar{s}_m <$
8803 $\bar{s}_{m \pm 1}$. Then search among these points to find the five sentences with the lowest
8804 \bar{s}_m .
8805 • How often do the five local minima correspond to paragraph boundaries?
 - 8806 – The fraction of local minima that are paragraph boundaries is the precision-at- k ,
8807 where in this case, $k = 5$.
 - 8808 – The fraction of paragraph boundaries which are local minima is the recall-at- k .
 - 8809 – Compute precision-at- k and recall-at- k for $k = 3$ and $k = 10$.

¹¹<http://homepages.inf.ed.ac.uk/mlap/coherence/>

- 8810 2. This exercise is to be done in pairs. Each participant selects an article from today's
8811 news, and replaces all mentions of individual people with special tokens like Person1,
8812 Person2, and so on. The other participant should then use the rules of centering
8813 theory to guess each type of referring expression: full name (Captain Ahab), partial
8814 name (e.g., Ahab), nominal (e.g., the ship's captain), or pronoun. Check whether
8815 the predictions match the original article, and whether the original article conforms
8816 to the rules of centering theory.
- 8817 3. In § 16.3.2.1, it is noted that bottom-up parsing with compositional representations
8818 of each span is not guaranteed to be optimal. In this exercise, you will construct
8819 a minimal example proving this point. Consider a discourse with four units, with
8820 base representations $\{z^{(i)}\}_{i=1}^4$. Construct a scenario in which the parse selected by
8821 bottom-up parsing is not optimal, and give the precise mathematical conditions that
8822 must hold for this suboptimal parse to be selected. You may ignore the relation labels
8823 ℓ for the purpose of this example.

8824

Part IV

8825

Applications

8826 Chapter 17

8827 Information extraction

8828 Computers offer powerful capabilities for searching and reasoning about structured records
8829 and relational data. Some even argue that the most important limitation of artificial
8830 intelligence is not inference or learning, but simply having too little knowledge (Lenat
8831 et al., 1990). Natural language processing provides an appealing solution: automatically
8832 construct a structured knowledge base by reading natural language text.

8833 For example, many Wikipedia pages have an “infobox” that provides structured information
8834 about an entity or event. An example is shown in Figure 17.1a: each row represents one
8835 or more properties of the entity In the Aeroplane Over the Sea, a record album. The set
8836 of properties is determined by a predefined schema, which applies to all record albums in
8837 Wikipedia. As shown in Figure 17.1b, the values for many of these fields are indicated
8838 directly in the first few sentences of text on the same Wikipedia page.

8839 The task of automatically constructing (or “populating”) an infobox from text is an
8840 example of information extraction. Much of information extraction can be described in
8841 terms of entities, relations, and events.

- 8842 • Entities are uniquely specified objects in the world, such as people (Jeff Mangum),
8843 places (Athens, Georgia), organizations (Merge Records), and times (February 10,
8844 1998). Chapter 8 described the task of named entity recognition, which labels tokens
8845 as parts of entity spans. Now we will see how to go further, linking each entity
8846 mention to an element in a knowledge base.
- 8847 • 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,...)
```

8848 The set of arguments for an event type is defined by a schema. Events often refer to
8849 time-delimited occurrences: weddings, protests, purchases, terrorist attacks.

8850 Information extraction is similar to semantic role labeling (chapter 13): we may think
8851 of predicates as corresponding to events, and the arguments as defining slots in the event
8852 representation. However, the goals of information extraction are different. Rather than
8853 accurately parsing every sentence, information extraction systems often focus on recognizing
8854 a few key relation or event types, or on the task of identifying all properties of a given entity.
8855 Information extraction is often evaluated by the correctness of the resulting knowledge base,
8856 and not by how many sentences were accurately parsed. The goal is sometimes described
8857 as macro-reading, as opposed to micro-reading, in which each sentence must be analyzed
8858 correctly. Macro-reading systems are not penalized for ignoring difficult sentences, as long
8859 as they can recover the same information from other, easier-to-read sources. However,
8860 macro-reading systems must resolve apparent inconsistencies (was the album released on
8861 Merge Records or Blue Rose Records?), requiring reasoning across the entire dataset.

8862 In addition to the basic tasks of recognizing entities, relations, and events, information
8863 extraction systems must handle negation, and must be able to distinguish statements of
8864 fact from hopes, fears, hunches, and hypotheticals. Finally, information extraction is often
8865 paired with the problem of question answering, which requires accurately parsing a query,
8866 and then selecting or generating a textual answer. Question answering systems can be
8867 built on knowledge bases that are extracted from large text corpora, or may attempt to
8868 identify answers directly from the source texts.

8869 17.1 Entities

8870 The starting point for information extraction is to identify mentions of entities in text.
8871 Consider the following example:

8872 (17.4) The United States Army captured a hill overlooking Atlanta on May 14, 1864.

8873 For this sentence, there are two goals:

- 8874 1. Identify the spans United States Army, Atlanta, and May 14, 1864 as entity mentions.
8875 (The hill is not uniquely identified, so it is not a named entity.) We may also want
8876 to recognize the named entity types: organization, location, and date. This is named
8877 entity recognition, and is described in chapter 8.
- 8878 2. Link these spans to entities in a knowledge base: U.S. Army, Atlanta, and 1864-May-
8879 14. This task is known as entity linking.

8880 The strings to be linked to entities are mentions — similar to the use of this term in
8881 coreference resolution. In some formulations of the entity linking task, only named entities
8882 are candidates for linking. This is sometimes called named entity linking (Ling et al., 2015).
8883 In other formulations, such as Wikification (Milne and Witten, 2008), any string can be a
8884 mention. The set of target entities often corresponds to Wikipedia pages, and Wikipedia is
8885 the basis for more comprehensive knowledge bases such as YAGO (Suchanek et al., 2007),
8886 DBpedia (Auer et al., 2007), and Freebase (Bollacker et al., 2008). Entity linking may also
8887 be performed in more “closed” settings, where a much smaller list of targets is provided in
8888 advance. The system must also determine if a mention does not refer to any entity in the
8889 knowledge base, sometimes called a NIL entity (McNamee and Dang, 2009).

8890 Returning to (17.4), the three entity mentions may seem unambiguous. But the
8891 Wikipedia disambiguation page for the string Atlanta says otherwise:¹ there are more
8892 than twenty different towns and cities, five United States Navy vessels, a magazine, a
8893 television show, a band, and a singer — each prominent enough to have its own Wikipedia

¹[https://en.wikipedia.org/wiki/Atlanta_\(disambiguation\)](https://en.wikipedia.org/wiki/Atlanta_(disambiguation)), retrieved November 1, 2017.

8894 page. We now consider how to choose among these dozens of possibilities. In this chapter
 8895 we will focus on supervised approaches. Unsupervised entity linking is closely related to
 8896 the problem of cross-document coreference resolution, where the task is to identify pairs
 8897 of mentions that corefer, across document boundaries (Bagga and Baldwin, 1998b; Singh
 8898 et al., 2011).

8899 17.1.1 Entity linking by learning to rank

8900 Entity linking is often formulated as a ranking problem,

$$\hat{y} = \operatorname{argmax}_{y \in \mathcal{Y}(\mathbf{x})} \Psi(y, \mathbf{x}, \mathbf{c}), \quad [17.1]$$

8901 where y is a target entity, \mathbf{x} is a description of the mention, $\mathcal{Y}(\mathbf{x})$ is a set of candidate
 8902 entities, and \mathbf{c} is a description of the context — such as the other text in the document,
 8903 or its metadata. The function Ψ is a scoring function, which could be a linear model,
 8904 $\Psi(y, \mathbf{x}, \mathbf{c}) = \boldsymbol{\theta} \cdot \mathbf{f}(y, \mathbf{x}, \mathbf{c})$, or a more complex function such as a neural network. In either
 8905 case, the scoring function can be learned by minimizing a margin-based ranking loss,

$$\ell(\hat{y}, y^{(i)}, \mathbf{x}^{(i)}, \mathbf{c}^{(i)}) = \left(\Psi(\hat{y}, \mathbf{x}^{(i)}, \mathbf{c}^{(i)}) - \Psi(y^{(i)}, \mathbf{x}^{(i)}, \mathbf{c}^{(i)}) + 1 \right)_+, \quad [17.2]$$

8906 where $y^{(i)}$ is the ground truth and $\hat{y} \neq y^{(i)}$ is the predicted target for mention $\mathbf{x}^{(i)}$ in
 8907 context $\mathbf{c}^{(i)}$ (Joachims, 2002; Dredze et al., 2010).

8908 Candidate identification For computational tractability, it is helpful to restrict the set
 8909 of candidates, $\mathcal{Y}(\mathbf{x})$. One approach is to use a name dictionary, which maps from strings
 8910 to the entities that they might mention. This mapping is many-to-many: a string such
 8911 as Atlanta can refer to multiple entities, and conversely, an entity such as Atlanta can be
 8912 referenced by multiple strings. A name dictionary can be extracted from Wikipedia, with
 8913 links between each Wikipedia entity page and the anchor text of all hyperlinks that point
 8914 to the page (Bunescu and Pasca, 2006; Ratinov et al., 2011). To improve recall, the name
 8915 dictionary can be augmented by partial and approximate matching (Dredze et al., 2010),
 8916 but as the set of candidates grows, the risk of false positives increases. For example, the
 8917 string Atlanta is a partial match to the Atlanta Fed (a name for the Federal Reserve Bank
 8918 of Atlanta), and a noisy match (edit distance of one) from Atalanta (a heroine in Greek
 8919 mythology and an Italian soccer team).

8920 Features Feature-based approaches to entity ranking rely on three main types of local
 8921 information (Dredze et al., 2010):

- 8922 • The similarity of the mention string to the canonical entity name, as quantified by
 8923 string similarity. This feature would elevate the city Atlanta over the basketball team
 8924 Atlanta Hawks for the string Atlanta.

- The popularity of the entity, which can be measured by Wikipedia page views or PageRank in the Wikipedia link graph. This feature would elevate Atlanta, Georgia over the unincorporated community of Atlanta, Ohio.
- The entity type, as output by the named entity recognition system. This feature would elevate the city of Atlanta over the magazine Atlanta in contexts where the mention is tagged as a location.

In addition to these local features, the document context can also help. If Jamaica is mentioned in a document about the Caribbean, it is likely to refer to the island nation; in the context of New York, it is likely to refer to the neighborhood in Queens; in the context of a menu, it might refer to a hibiscus tea beverage. Such hints can be formalized by computing the similarity between the Wikipedia page describing each candidate entity and the mention context $\mathbf{c}^{(i)}$, which may include the bag-of-words representing the document (Dredze et al., 2010; Hoffart et al., 2011) or a smaller window of text around the mention (Ratinov et al., 2011). For example, we can compute the cosine similarity between bag-of-words vectors for the context and entity description, typically weighted using inverse document frequency to emphasize rare words.²

Neural entity linking An alternative approach is to compute the score for each entity candidate using distributed vector representations of the entities, mentions, and context. For example, for the task of entity linking in Twitter, Yang et al. (2016) employ the bilinear scoring function,

$$\Psi(y, \mathbf{x}, \mathbf{c}) = \mathbf{v}_y^\top \Theta^{(y,x)} \mathbf{x} + \mathbf{v}_y^\top \Theta^{(y,c)} \mathbf{c}, \quad [17.3]$$

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 mention, $\mathbf{c} \in \mathbb{R}^{K_c}$ as the embedding of the context, and the matrices $\Theta^{(y,x)}$ and $\Theta^{(y,c)}$ as parameters that score the compatibility of each entity with respect to the mention and context. Each of the vector embeddings can be learned from an end-to-end objective, or pre-trained on unlabeled data.

- Pretrained entity embeddings can be obtained from an existing knowledge base (Bordes et al., 2011, 2013), or by running a word embedding algorithm such as word2vec on the text of Wikipedia, with hyperlinks substituted for the anchor text.³

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).

³Pre-trained entity embeddings can be downloaded from <https://code.google.com/archive/p/word2vec/>.

- 8953 • The embedding of the mention \mathbf{x} can be computed by averaging the embeddings of
 8954 the words in the mention (Yang et al., 2016), or by the compositional techniques
 8955 described in § 14.8.
- 8956 • The embedding of the context \mathbf{c} can also be computed from the embeddings of the
 8957 words in the context. A denoising autoencoder learns a function from raw text to
 8958 dense K -dimensional vector encodings by minimizing a reconstruction loss (Vincent
 8959 et al., 2010),

$$\min_{\boldsymbol{\theta}_g, \boldsymbol{\theta}_h} \sum_{i=1}^N \|\mathbf{x}^{(i)} - g(h(\tilde{\mathbf{x}}^{(i)}; \boldsymbol{\theta}_h); \boldsymbol{\theta}_g)\|^2, \quad [17.4]$$

8960 where $\tilde{\mathbf{x}}^{(i)}$ is a noisy version of the bag-of-words counts $\mathbf{x}^{(i)}$, which is produced by
 8961 randomly setting some counts to zero; $h : \mathbb{R}^V \mapsto \mathbb{R}^K$ is an encoder with parameters
 8962 $\boldsymbol{\theta}_h$; and $g : \mathbb{R}^K \mapsto \mathbb{R}^V$, with parameters $\boldsymbol{\theta}_g$. The encoder and decoder functions
 8963 are typically implemented as feedforward neural networks. To apply this model to
 8964 entity linking, each entity and context are initially represented by the encoding of
 8965 their bag-of-words vectors, $h(\mathbf{e})$ and $g(\mathbf{c})$, and these encodings are then fine-tuned
 8966 from labeled data (He et al., 2013). The context vector \mathbf{c} can also be obtained by
 8967 convolution on the embeddings of words in the document (Sun et al., 2015), or by
 8968 examining metadata such as the author’s social network (Yang et al., 2016).

8969 The remaining parameters $\Theta^{(y,x)}$ and $\Theta^{(y,c)}$ can be trained by backpropagation from the
 8970 margin loss in Equation 17.2.

8971 17.1.2 Collective entity linking

8972 Entity linking can be more accurate when it is performed jointly across a document. To
 8973 see why, consider the following lists:

- 8974 (17.5) California, Oregon, Washington
 8975 (17.6) Baltimore, Washington, Philadelphia
 8976 (17.7) Washington, Adams, Jefferson

8977 In each case, the term Washington refers to a different entity, and this reference is strongly
 8978 suggested by the other entries on the list. In the last list, all three names are highly
 8979 ambiguous — there are dozens of other Adams and Jefferson entities in Wikipedia. But
 8980 a preference for coherence motivates collectively linking these references to the first three
 8981 U.S. presidents.

8978 A general approach to collective entity linking is to introduce a compatibility score
 8979 $\psi_c(\mathbf{y})$. Collective entity linking is then performed by optimizing the global objective,

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathbb{Y}(\mathbf{x})}{\operatorname{argmax}} \Psi_c(\mathbf{y}) + \sum_{i=1}^N \Psi_\ell(y^{(i)}, \mathbf{x}^{(i)}, \mathbf{c}^{(i)}), \quad [17.5]$$

8980 where $\mathbb{Y}(\mathbf{x})$ is the set of all possible collective entity assignments for the mentions in \mathbf{x} , and
 8981 ψ_ℓ is the local scoring function for each entity i . The compatibility function is typically
 8982 decomposed into a sum of pairwise scores, $\Psi_c(\mathbf{y}) = \sum_{i=1}^N \sum_{j \neq i} \Psi_c(y^{(i)}, y^{(j)})$. These scores
 8983 can be computed in a number of different ways:

- 8984 • Wikipedia defines high-level categories for entities (e.g., living people, Presidents of
 8985 the United States, States of the United States), and Ψ_c can reward entity pairs for
 8986 the number of categories that they have in common (Cucerzan, 2007).
- 8987 • Compatibility can be measured by the number of incoming hyperlinks shared by the
 8988 Wikipedia pages for the two entities (Milne and Witten, 2008).
- 8989 • In a neural architecture, the compatibility of two entities can be set equal to the
 8990 inner product of their embeddings, $\Psi_c(y^{(i)}, y^{(j)}) = \mathbf{v}_{y^{(i)}} \cdot \mathbf{v}_{y^{(j)}}$.
- 8991 • A non-pairwise compatibility score can be defined using a type of latent variable
 8992 model known as a probabilistic topic model (Blei et al., 2003; Blei, 2012). In this
 8993 framework, each latent topic is a probability distribution over entities, and each
 8994 document has a probability distribution over topics. Each entity helps to determine
 8995 the document's distribution over topics, and in turn these topics help to resolve
 8996 ambiguous entity mentions (Newman et al., 2006). Inference can be performed using
 8997 the sampling techniques described in chapter 5.

8998 Unfortunately, collective entity linking is NP-hard even for pairwise compatibility functions,
 8999 so exact optimization is almost certainly intractable. Various approximate inference techniques
 9000 have been proposed, including integer linear programming (Cheng and Roth, 2013), Gibbs
 9001 sampling (Han and Sun, 2012), and graph-based algorithms (Hoffart et al., 2011; Han et al.,
 9002 2011).

9003 17.1.3 *Pairwise ranking loss functions

9004 The loss function defined in Equation 17.2 considers only the highest-scoring prediction \hat{y} ,
 9005 but in fact, the true entity $y^{(i)}$ should outscore all other entities. A loss function based on
 9006 this idea would give a gradient against the features or representations of several entities,
 9007 not just the top-scoring prediction. Usunier et al. (2009) define a general ranking error
 9008 function,

$$L_{\text{rank}}(k) = \sum_{j=1}^k \alpha_j, \quad \text{with } \alpha_1 \geq \alpha_2 \geq \dots \geq 0, \quad [17.6]$$

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
  
```

9009 where k is equal to the number of labels ranked higher than the correct label $y^{(i)}$. This
 9010 function defines a class of ranking errors: if $\alpha_j = 1$ for all j , then the ranking error is
 9011 equal to the rank of the correct entity; if $\alpha_1 = 1$ and $\alpha_{j>1} = 0$, then the ranking error is
 9012 one whenever the correct entity is not ranked first; if α_j decreases smoothly with j , as in
 9013 $\alpha_j = \frac{1}{j}$, then the error is between these two extremes.

This ranking error can be integrated into a margin objective. Remember that large margin classification requires not only the correct label, but also that the correct label outscores other labels by a substantial margin. A similar principle applies to ranking: we want a high rank for the correct entity, and we want it to be separated from other entities by a substantial margin. We therefore define the margin-augmented rank,

$$r(y^{(i)}, \mathbf{x}^{(i)}) \triangleq \sum_{y \in \mathcal{Y}(\mathbf{x}^{(i)}) \setminus y^{(i)}} \delta\left(1 + \psi(y, \mathbf{x}^{(i)}) \geq \psi(y^{(i)}, \mathbf{x}^{(i)})\right), \quad [17.7]$$

9014 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
 9015 true entity $y^{(i)}$. The margin-augmented rank is the rank of the true entity, after augmenting
 9016 every other candidate with a margin of one, under the current scoring function ψ . (The
 9017 context \mathbf{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,

$$\ell(y^{(i)}, \mathbf{x}^{(i)}) = \frac{L_{\text{rank}}(r(y^{(i)}, \mathbf{x}^{(i)}))}{r(y^{(i)}, \mathbf{x}^{(i)})} \sum_{y \in \mathcal{Y}(\mathbf{x}) \setminus y^{(i)}} \left(\psi(y, \mathbf{x}^{(i)}) - \psi(y^{(i)}, \mathbf{x}^{(i)}) + 1 \right)_+, \quad [17.8]$$

9018 The sum in Equation 17.8 includes non-zero values for every label that is ranked at least
 9019 as high as the true entity, after applying the margin augmentation. Dividing by the
 9020 margin-augmented rank of the true entity thus gives the average violation.

9021 The objective in Equation 17.8 is expensive to optimize when the label space is large,
 9022 as is usually the case for entity linking against large knowledge bases. This motivates a
 9023 randomized approximation called WARP (Weston et al., 2011), shown in Algorithm 19. In
 9024 this procedure, we sample random entities until one violates the pairwise margin constraint,
 9025 $\psi(y, \mathbf{x}^{(i)}) + 1 \geq \psi(y^{(i)}, \mathbf{x}^{(i)})$. The number of samples N required to find such a violation
 9026 yields an approximation of the margin-augmented rank of the true entity, $r(y^{(i)}, \mathbf{x}^{(i)}) \approx$
 9027 $\left\lfloor \frac{|\mathcal{Y}(\mathbf{x})|}{N} \right\rfloor$. If a violation is found immediately, $N = 1$, the correct entity probably ranks
 9028 below many others, $r \approx |\mathcal{Y}(\mathbf{x})|$. If many samples are required before a violation is found,
 9029 $N \rightarrow |\mathcal{Y}(\mathbf{x})|$, then the correct entity is probably highly ranked, $r \rightarrow 1$. A computational
 9030 advantage of WARP is that it is not necessary to find the highest-scoring label, which can
 9031 impose a non-trivial computational cost when $\mathcal{Y}(\mathbf{x}^{(i)})$ is large. The objective is conceptually
 9032 similar to the negative sampling objective in word2vec (chapter 14), which compares the
 9033 observed word against randomly sampled alternatives.

9034 17.2 Relations

9035 After identifying the entities that are mentioned in a text, the next step is to determine
 9036 how they are related. Consider the following example:

9037 (17.8) George Bush traveled to France on Thursday for a summit.

9038 This sentence introduces a relation between the entities referenced by George Bush and
 9039 France. In the Automatic Content Extraction (ACE) ontology (Linguistic Data Consortium,
 9040 2005), the type of this relation is physical, and the subtype is located. This relation would
 9041 be written,

$$\text{Physical.Located(George Bush, France).} \quad [17.9]$$

9042 Relations take exactly two arguments, and the order of the arguments matters.

9043 In the ACE datasets, relations are annotated between entity mentions, as in the example
 9044 above. Relations can also hold between nominals, as in the following example from the
 9045 SemEval-2010 shared task (Hendrickx et al., 2009):

9046 (17.9) The cup contained tea from dried ginseng.

9047 This sentence describes a relation of type entity-origin between tea and ginseng. Nominal
 9048 relation extraction is closely related to semantic role labeling (chapter 13). The main
 9049 difference is that relation extraction is restricted to a relatively small number of relation
 9050 types; for example, Table 17.1 shows the ten relation types from SemEval-2010.

cause-effect	those cancers were caused by radiation exposures
instrument-agency	phone operator
product-producer	a factory manufactures suits
content-container	a bottle of honey was weighed
entity-origin	letters from foreign countries
entity-destination	the boy went to bed
component-whole	my apartment has a large kitchen
member-collection	there are many trees in the forest
communication-topic	the lecture was about semantics

Table 17.1: Relations and example sentences from the SemEval-2010 dataset (Hendrickx et al., 2009)

9051 17.2.1 Pattern-based relation extraction

9052 Early work on relation extraction focused on hand-crafted patterns (Hearst, 1992). For
 9053 example, the appositive Starbuck, a native of Nantucket signals the relation entity-origin
 9054 between Starbuck and Nantucket. This pattern can be written as,

$$\text{Person , a native of Location} \Rightarrow \text{entity-origin(Person, Location)}. \quad [17.10]$$

9055 This pattern will be “triggered” whenever the literal string , a native of occurs between
 9056 an entity of type Person and an entity of type Location. Such patterns can be generalized
 9057 beyond literal matches using techniques such as lemmatization, which would enable the
 9058 words (buy, buys, buying) to trigger the same patterns (see § 4.3.1.2). A more aggressive
 9059 strategy would be to group all words in a WordNet synset (§ 4.2), so that, e.g., buy and
 9060 purchase trigger the same patterns.

9061 Relation extraction patterns can be implemented in finite-state automata (§ 9.1). If the
 9062 named entity recognizer is also a finite-state machine, then the systems can be combined
 9063 by finite-state transduction (Hobbs et al., 1997). This makes it possible to propagate
 9064 uncertainty through the finite-state cascade, and disambiguate from higher-level context.
 9065 For example, suppose the entity recognizer cannot decide whether Starbuck refers to either
 9066 a Person or a Location; in the composed transducer, the relation extractor would be free
 9067 to select the Person annotation when it appears in the context of an appropriate pattern.

9068 17.2.2 Relation extraction as a classification task

9069 Relation extraction can be formulated as a classification problem,

$$\hat{r}_{(i,j),(m,n)} = \underset{r \in \mathcal{R}}{\operatorname{argmax}} \Psi(r, (i, j), (m, n), \mathbf{w}), \quad [17.11]$$

where $r \in \mathcal{R}$ is a relation type (possibly nil), $\mathbf{w}_{i+1:j}$ is the span of the first argument, and $\mathbf{w}_{m+1:n}$ is the span of the second argument. The argument $\mathbf{w}_{m+1:n}$ may appear before or after $\mathbf{w}_{i+1:j}$ in the text, or they may overlap; we stipulate only that $\mathbf{w}_{i+1:j}$ is the first argument of the relation. We now consider three alternatives for computing the scoring function.

17.2.2.1 Feature-based classification

In a feature-based classifier, the scoring function is defined as,

$$\Psi(r, (i, j), (m, n), \mathbf{w}) = \boldsymbol{\theta} \cdot \mathbf{f}(r, (i, j), (m, n), \mathbf{w}), \quad [17.12]$$

with $\boldsymbol{\theta}$ representing a vector of weights, and $\mathbf{f}(\cdot)$ a vector of features. The pattern-based methods described in § 17.2.1 suggest several features:

- Local features of $\mathbf{w}_{i+1:j}$ and $\mathbf{w}_{m+1:n}$, including: the strings themselves; whether they are recognized as entities, and if so, which type; whether the strings are present in a gazetteer of entity names; each string's syntactic head (§ 9.2.2).
- Features of the span between the two arguments, $\mathbf{w}_{j+1:m}$ or $\mathbf{w}_{n+1:i}$ (depending on which argument appears first): the length of the span; the specific words that appear in the span, either as a literal sequence or a bag-of-words; the wordnet synsets (§ 4.2) that appear in the span between the arguments.
- Features of the syntactic relationship between the two arguments, typically the dependency path between the arguments (§ 13.2.1). Example dependency paths are shown in Table 17.2.

17.2.2.2 Kernels

Suppose that the first line of Table 17.2 is a labeled example, and the remaining lines are instances to be classified. A feature-based approach would have to decompose the dependency paths into features that capture individual edges, with or without their labels, and then learn weights for each of these features: for example, the second line contains identical dependencies, but different arguments; the third line contains a different inflection of the word travel; the fourth and fifth lines each contain an additional edge on the dependency path; and the sixth example uses an entirely different path. Rather than attempting to create local features that capture all of the ways in which these dependencies paths are similar and different, we can instead define a similarity function κ , which computes a score for any pair of instances, $\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}_+$. The score for any pair of instances (i, j) 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 function κ obeys a few key properties it is a valid kernel function.⁴

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 ($\forall \mathbf{a}, \mathbf{a}^\top \mathbf{K} \mathbf{a} \geq 0$).

1. George Bush traveled to France	George Bush \leftarrow traveled \rightarrow France Nsubj Obl
2. Ahab traveled to Nantucket	Ahab \leftarrow traveled \rightarrow Nantucket Nsubj obl
3. George Bush will travel to France	George Bush \leftarrow travel \rightarrow France Nsubj Obl
4. George Bush wants to travel to France	George Bush \leftarrow wants \rightarrow travel \rightarrow France Nsubj Xcomp Obl
5. Ahab traveled to a city in France	Ahab \leftarrow traveled \rightarrow city \rightarrow France Nsubj Obl Nmod
6. We await Ahab's visit to France	Ahab \leftarrow visit \rightarrow France Nmod:poss Nmod

Table 17.2: Candidates instances for the Physical.Located relation, and their dependency paths

Given a valid kernel function, we can build a non-linear classifier without explicitly defining a feature vector or neural network architecture. For a binary classification problem $y \in \{-1, 1\}$, we have the decision function,

$$\hat{y} = \text{Sign}(b + \sum_{i=1}^N y^{(i)} \alpha^{(i)} \kappa(\mathbf{x}^{(i)}, \mathbf{x})) \quad [17.13]$$

where b and $\{\alpha^{(i)}\}_{i=1}^N$ are parameters that must be learned from the training set, under the constraint $\forall_i, \alpha^{(i)} \geq 0$. Intuitively, each α_i specifies the importance of the instance $\mathbf{x}^{(i)}$ towards the classification rule. Kernel-based classification can be viewed as a weighted form of the nearest-neighbor classifier (Hastie et al., 2009), in which test instances are assigned the most common label among their near neighbors in the training set. This results in a non-linear classification boundary. The parameters are typically learned from a margin-based objective (see § 2.3), leading to the kernel support vector machine. To generalize to multi-class classification, we can train separate binary classifiers for each label (sometimes called one-versus-all), or train binary classifiers for each pair of possible labels (one-versus-one).

Dependency kernels are particularly effective for relation extraction, due to their ability to capture syntactic properties of the path between the two candidate arguments. One class of dependency tree kernels is defined recursively, with the score for a pair of trees equal to the similarity of the root nodes and the sum of similarities of matched pairs of child subtrees (Zelenko et al., 2003; Culotta and Sorensen, 2004). Alternatively, Bunescu and Mooney (2005) define a kernel function over sequences of unlabeled dependency edges, in which the score is computed as a product of scores for each pair of words in the sequence: identical words receive a high score, words that share a synset or part-of-speech receive a small non-zero score (e.g., travel / visit), and unrelated words receive a score of zero.

0). For more on kernel-based classification, see chapter 14 of Murphy (2012).

9121 17.2.2.3 Neural relation extraction

9122 Convolutional neural networks were an early neural architecture for relation extraction (Zeng
 9123 et al., 2014; dos Santos et al., 2015). For the sentence (w_1, w_2, \dots, w_M) , obtain a matrix
 9124 of word embeddings \mathbf{X} , where $\mathbf{x}_m \in \mathbb{R}^K$ is the embedding of w_m . Now, suppose the
 9125 candidate arguments appear at positions a_1 and a_2 ; then for each word in the sentence, its
 9126 position with respect to each argument is $m - a_1$ and $m - a_2$. (Following Zeng et al. (2014),
 9127 this is a restricted version of the relation extraction task in which the arguments are single
 9128 tokens.) To capture any information conveyed by these positions, the word embeddings are
 9129 concatenated with embeddings of the positional offsets, $\mathbf{x}_{m-a_1}^{(p)}$ and $\mathbf{x}_{m-a_2}^{(p)}$. The complete
 9130 base representation of the sentence is,

$$\mathbf{X}(a_1, a_2) = \begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_M \\ \mathbf{x}_{1-a_1}^{(p)} & \mathbf{x}_{2-a_1}^{(p)} & \cdots & \mathbf{x}_{M-a_1}^{(p)} \\ \mathbf{x}_{1-a_2}^{(p)} & \mathbf{x}_{2-a_2}^{(p)} & \cdots & \mathbf{x}_{M-a_2}^{(p)} \end{pmatrix}, \quad [17.14]$$

9131 where each column is a vertical concatenation of a word embedding, represented by the
 9132 column vector \mathbf{x}_m , and two positional embeddings, specifying the position with respect to
 9133 a_1 and a_2 . The matrix $\mathbf{X}(a_1, a_2)$ is then taken as input to a convolutional layer (see § 3.4),
 9134 and max-pooling is applied to obtain a vector. The final scoring function is then,

$$\Psi(r, i, j, \mathbf{X}) = \theta_r \cdot \text{MaxPool}(\text{ConvNet}(\mathbf{X}(i, j); \phi)), \quad [17.15]$$

where ϕ defines the parameters of the convolutional operator, and the θ_r defines a set of
 weights for relation r . The model can be trained using a margin objective,

$$\hat{r} = \underset{r}{\operatorname{argmax}} \Psi(r, i, j, \mathbf{X}) \quad [17.16]$$

$$\ell = (1 + \psi(\hat{r}, i, j, \mathbf{X}) - \psi(r, i, j, \mathbf{X}))_+. \quad [17.17]$$

9135 Recurrent neural networks have also been applied to relation extraction, using a network
 9136 such as an bidirectional LSTM to encode the words or dependency path between the two
 9137 arguments. Xu et al. (2015) segment each dependency path into left and right subpaths:
 9138 the path George Bush $\xleftarrow{\text{Nsubj}}$ wants $\xrightarrow{\text{Xcomp}}$ travel $\xrightarrow{\text{Obl}}$ France is segmented into the subpaths,

9139 (17.10) George Bush $\xleftarrow{\text{Nsubj}}$ wants

9140 (17.11) wants $\xrightarrow{\text{Xcomp}}$ travel $\xrightarrow{\text{Obl}}$ France.

Xu et al. (2015) then run recurrent networks from the arguments to the root word (in this
 case, wants), obtaining the final representation by max pooling across all the recurrent
 states along each path. This process can be applied across separate “channels”, in which

the inputs consist of embeddings for the words, parts-of-speech, dependency relations, and WordNet hypernyms. To define the model formally, let $s(m)$ define the successor of word m in either the left or right subpath (in a dependency path, each word can have a successor in at most one subpath). Let $\mathbf{x}_m^{(c)}$ indicate the embedding of word (or relation) m in channel c , and let $\overleftarrow{\mathbf{h}}_m^{(c)}$ and $\overrightarrow{\mathbf{h}}_m^{(c)}$ indicate the associated recurrent states in the left and right subtrees respectively. Then the complete model is specified as follows,

$$\mathbf{h}_{s(m)}^{(c)} = \text{RNN}(\mathbf{x}_{s(m)}^{(c)}, \mathbf{h}_m^{(c)}) \quad [17.18]$$

$$\mathbf{z}^{(c)} = \text{MaxPool} \left(\overleftarrow{\mathbf{h}}_i^{(c)}, \overleftarrow{\mathbf{h}}_{s(i)}^{(c)}, \dots, \overleftarrow{\mathbf{h}}_{\text{root}}^{(c)}, \overrightarrow{\mathbf{h}}_j^{(c)}, \overrightarrow{\mathbf{h}}_{s(j)}^{(c)}, \dots, \overrightarrow{\mathbf{h}}_{\text{root}}^{(c)} \right) \quad [17.19]$$

$$\Psi(r, i, j) = \theta \cdot [\mathbf{z}^{(\text{word})}; \mathbf{z}^{(\text{POS})}; \mathbf{z}^{(\text{dependency})}; \mathbf{z}^{(\text{hypernym})}] . \quad [17.20]$$

9141 Note that \mathbf{z} is computed by applying max-pooling to the matrix of horizontally concatenated
 9142 vectors \mathbf{h} , while Ψ is computed from the vector of vertically concatenated vectors \mathbf{z} . Xu
 9143 et al. (2015) pass the score Ψ through a softmax layer to obtain a probability $p(r | i, j, \mathbf{w})$,
 9144 and train the model by regularized cross-entropy. Miwa and Bansal (2016) show that
 9145 a related model can solve the more challenging “end-to-end” relation extraction task, in
 9146 which the model must simultaneously detect entities and then extract their relations.

9147 17.2.3 Knowledge base population

9148 In many applications, what matters is not what fraction of sentences are analyzed correctly,
 9149 but how much accurate knowledge can be extracted. Knowledge base population (KBP)
 9150 refers to the task of filling in Wikipedia-style infoboxes, as shown in Figure 17.1a. Knowledge
 9151 base population can be decomposed into two subtasks: entity linking (described in § 17.1),
 9152 and slot filling (Ji and Grishman, 2011). Slot filling has two key differences from the
 9153 formulation of relation extraction presented above: the relations hold between entities
 9154 rather than spans of text, and the performance is evaluated at the type level (on entity
 9155 pairs), rather than on the token level (on individual sentences).

9156 From a practical standpoint, there are three other important differences between slot
 9157 filling and per-sentence relation extraction.

- 9158 • KBP tasks are often formulated from the perspective of identifying attributes of a few
 9159 “query” entities. As a result, these systems often start with an information retrieval
 9160 phase, in which relevant passages of text are obtained by search.
- 9161 • For many entity pairs, there will be multiple passages of text that provide evidence.
 9162 Slot filling systems must aggregate this evidence to predict a single relation type (or
 9163 set of relations).
- 9164 • Labeled data is usually available in the form of pairs of related entities, rather than
 9165 annotated passages of text. Training from such type-level annotations is a challenge:

9166 two entities may be linked by several relations, or they may appear together in a
 9167 passage of text that nonetheless does not describe their relation to each other.

9168 Information retrieval is beyond the scope of this text (see Manning et al., 2008). The
 9169 remainder of this section describes approaches to information fusion and learning from
 9170 type-level annotations.

9171 17.2.3.1 Information fusion

9172 In knowledge base population, there will often be multiple pieces of evidence for (and
 9173 sometimes against) a single relation. For example, a search for the entity Maynard Jackson,
 9174 Jr. may return several passages that reference the entity Atlanta:⁵

9175 (17.12) Elected mayor of Atlanta in 1973, Maynard Jackson was the first African American
 9176 to serve as mayor of a major southern city.

9177 (17.13) Atlanta's airport will be renamed to honor Maynard Jackson, the city's first Black
 9178 mayor .

9179 (17.14) Born in Dallas, Texas in 1938, Maynard Holbrook Jackson, Jr. moved to Atlanta
 9180 when he was 8.

9181 (17.15) Maynard Jackson has gone from one of the worst high schools in Atlanta to one
 9182 of the best.

9183 The first and second examples provide evidence for the relation mayor holding between
 9184 the entities Atlanta and Maynard Jackson, Jr.. The third example provides evidence for a
 9185 different relation between these same entities, lived-in. The fourth example poses an entity
 9186 linking problem, referring to Maynard Jackson high school. Knowledge base population
 9187 requires aggregating this sort of textual evidence, and predicting the relations that are
 9188 most likely to hold.

9189 One approach is to run a single-document relation extraction system (using the techniques
 9190 described in § 17.2.2), and then aggregate the results (Li et al., 2011). Relations that are
 9191 detected with high confidence in multiple documents are more likely to be valid, motivating
 9192 the heuristic,

$$\psi(r, e_1, e_2) = \sum_{i=1}^N (\text{p}(r(e_1, e_2) | \mathbf{w}^{(i)}))^\alpha, \quad [17.21]$$

9193 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
 9194 on the text $\mathbf{w}^{(i)}$, and $\alpha \gg 1$ is a tunable hyperparameter. Using this heuristic, it is possible

⁵First three examples from: <http://www.georgiaencyclopedia.org/articles/government-politics/maynard-jackson-1938-2003>; JET magazine, November 10, 2003; www.todayingeorgiahistory.org/content/maynard-jackson-elected

9195 to rank all candidate relations, and trace out a precision-recall curve as more relations are
 9196 extracted.⁶ Alternatively, features can be aggregated across multiple passages of text,
 9197 feeding a single type-level relation extraction system (Wolfe et al., 2017).

9198 Precision can be improved by introducing constraints across multiple relations. For
 9199 example, if we are certain of the relation $\text{parent}(e_1, e_2)$, then it cannot also be the case
 9200 that $\text{parent}(e_2, e_1)$. Integer linear programming makes it possible to incorporate such
 9201 constraints into a global optimization (Li et al., 2011). Other pairs of relations have
 9202 positive correlations, such $\text{mayor}(e_1, e_2)$ and $\text{lived-in}(e_1, e_2)$. Compatibility across relation
 9203 types can be incorporated into probabilistic graphical models (e.g., Riedel et al., 2010).

9204 17.2.3.2 Distant supervision

9205 Relation extraction is “annotation hungry,” because each relation requires its own labeled
 9206 data. Rather than relying on annotations of individual documents, it would be preferable
 9207 to use existing knowledge resources — such as the many facts that are already captured
 9208 in knowledge bases like DBPedia. However such annotations raise the inverse of the
 9209 information fusion problem considered above: the existence of the relation $\text{mayor}(\text{Maynard Jackson Jr., Atlanta},$
 9210 provides only distant supervision for the example texts in which this entity pair is mentioned.

9211 One approach is to treat the entity pair as the instance, rather than the text itself (Mintz
 9212 et al., 2009). Features are then aggregated across all sentences in which both entities are
 9213 mentioned, and labels correspond to the relation (if any) between the entities in a knowledge
 9214 base, such as FreeBase. Negative instances are constructed from entity pairs that are not
 9215 related in the knowledge base. In some cases, two entities are related, but the knowledge
 9216 base is missing the relation; however, because the number of possible entity pairs is huge,
 9217 these missing relations are presumed to be relatively rare. This approach is shown in
 9218 Figure 17.2.

9219 In multiple instance learning, labels are assigned to sets of instances, of which only an
 9220 unknown subset are actually relevant (Dietterich et al., 1997; Maron and Lozano-Pérez,
 9221 1998). This formalizes the framework of distant supervision: the relation $\text{rel}(a, b)$ acts as
 9222 a label for the entire set of sentences mentioning entities a and b , even when only a subset
 9223 of these sentences actually describes the relation. One approach to multi-instance learning
 9224 is to introduce a binary latent variable for each sentence, indicating whether the sentence
 9225 expresses the labeled relation (Riedel et al., 2010). A variety of inference techniques have
 9226 been employed for this probabilistic model of relation extraction: Surdeanu et al. (2012) use
 9227 expectation maximization, Riedel et al. (2010) use sampling, and Hoffmann et al. (2011)
 9228 use a custom graph-based algorithm. Expectation maximization and sampling are surveyed
 9229 in chapter 5, and are covered in more detail by Murphy (2012); graph-based methods are
 9230 surveyed by Mihalcea and Radev (2011).

⁶The precision-recall curve is similar to the ROC curve shown in Figure 4.4, but it includes the precision $\frac{\text{TP}}{\text{TP} + \text{FP}}$ rather than the false positive rate $\frac{\text{FP}}{\text{FP} + \text{TN}}$.

- Label : mayor(Atlanta, Maynard Jackson)
 - Elected mayor of Atlanta in 1973, Maynard Jackson ...
 - Atlanta's airport will be renamed to honor Maynard Jackson, the city's first Black mayor
 - Born in Dallas, Texas in 1938, Maynard Holbrook Jackson, Jr. moved to Atlanta when he was 8.
- Label : mayor(New York, Fiorello La Guardia)
 - Fiorello La Guardia was Mayor of New York for three terms ...
 - Fiorello La Guardia, then serving on the New York City Board of Aldermen...
- Label : born-in(Dallas, Maynard Jackson)
 - Born in Dallas, Texas in 1938, Maynard Holbrook Jackson, Jr. moved to Atlanta when he was 8.
 - Maynard Jackson was raised in Dallas ...
- Label : nil(New York, Maynard Jackson)
 - Jackson married Valerie Richardson, whom he had met in New York...
 - Jackson was a member of the Georgia and New York bars ...

Figure 17.2: Four training instances for relation classification using distant supervision Mintz et al. (2009). The first two instances are positive for the mayor relation, and the third instance is positive for the born-in relation. The fourth instance is a negative example, constructed from a pair of entities (New York, Maynard Jackson) that do not appear in any Freebase relation. Each instance's features are computed by aggregating across all sentences in which the two entities are mentioned.

9231 17.2.4 Open information extraction

9232 In classical relation extraction, the set of relations is defined in advance, using a schema.
 9233 The relation for any pair of entities can then be predicted using multi-class classification.
 9234 In open information extraction (OpenIE), a relation can be any triple of text. The example
 9235 sentence (17.12) instantiates several “relations” of this sort:

- 9236 • (mayor of, Maynard Jackson, Atlanta),
 9237 • (elected, Maynard Jackson, mayor of Atlanta),
 9238 • (elected in, Maynard Jackson, 1973),

9239 and so on. Extracting such tuples can be viewed as a lightweight version of semantic role
 9240 labeling (chapter 13), with only two argument types: first slot and second slot. The task is
 9241 generally evaluated on the relation level, rather than on the level of sentences: precision is
 9242 measured by the number of extracted relations that are accurate, and recall is measured by

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.

9243 the number of true relations that were successfully extracted. OpenIE systems are trained
 9244 from distant supervision or bootstrapping, rather than from labeled sentences.

9245 An early example is the TextRunner system (Banko et al., 2007), which identifies
 9246 relations with a set of handcrafted syntactic rules. The examples that are acquired from
 9247 the handcrafted rules are then used to train a classification model that uses part-of-speech
 9248 patterns as features. Finally, the relations that are extracted by the classifier are aggregated,
 9249 removing redundant relations and computing the number of times that each relation is
 9250 mentioned in the corpus. TextRunner was the first in a series of systems that performed
 9251 increasingly accurate open relation extraction by incorporating more precise linguistic
 9252 features (Etzioni et al., 2011), distant supervision from Wikipedia infoboxes (Wu and
 9253 Weld, 2010), and better learning algorithms (Zhu et al., 2009).

9254 17.3 Events

9255 Relations link pairs of entities, but many real-world situations involve more than two
 9256 entities. Consider again the example sentence (17.12), which describes the event of an
 9257 election, with four properties: the office (mayor), the district (Atlanta), the date (1973),
 9258 and the person elected (Maynard Jackson, Jr.). In event detection, a schema is provided
 9259 for each event type (e.g., an election, a terrorist attack, or a chemical reaction), indicating
 9260 all the possible properties of the event. The system is then required to fill in as many of
 9261 these properties as possible (Doddington et al., 2004).

9262 Event detection systems generally involve a retrieval component (finding relevant documents
 9263 and passages of text) and an extraction component (determining the properties of the
 9264 event based on the retrieved texts). Early approaches focused on finite-state patterns for
 9265 identifying event properties (Hobbs et al., 1997); such patterns can be automatically induced
 9266 by searching for patterns that are especially likely to appear in documents that match the
 9267 event query (Riloff, 1996). Contemporary approaches employ techniques that are similar

9268 to FrameNet semantic role labeling (§ 13.2), such as structured prediction over local and
9269 global features (Li et al., 2013) and bidirectional recurrent neural networks (Feng et al.,
9270 2016). These methods detect whether an event is described in a sentence, and if so, what
9271 are its properties.

9272 Event coreference Because multiple sentences may describe unique properties of a single
9273 event, event coreference is required to link event mentions across a single passage of
9274 text, or between passages (Humphreys et al., 1997). Bejan and Harabagiu (2014) define
9275 event coreference as the task of identifying event mentions that share the same event
9276 participants (i.e., the slot-filling entities) and the same event properties (e.g., the time
9277 and location), within or across documents. Event coreference resolution can be performed
9278 using supervised learning techniques in a similar way to entity coreference, as described
9279 in chapter 15: move left-to-right through the document, and use a classifier to decide
9280 whether to link each event reference to an existing cluster of coreferent events, or to
9281 create a new cluster (Ahn, 2006). Each clustering decision is based on the compatibility
9282 of features describing the participants and properties of the event. Due to the difficulty
9283 of annotating large amounts of data for entity coreference, unsupervised approaches are
9284 especially desirable (Chen and Ji, 2009; Bejan and Harabagiu, 2014).

9285 Relations between events Just as entities are related to other entities, events may be
9286 related to other events: for example, the event of winning an election both precedes and
9287 causes the event of serving as mayor; moving to Atlanta precedes and enables the event
9288 of becoming mayor of Atlanta; moving from Dallas to Atlanta prevents the event of later
9289 becoming mayor of Dallas. As these examples show, events may be related both temporally
9290 and causally. The TimeML annotation scheme specifies a set of six temporal relations
9291 between events (Pustejovsky et al., 2005), derived in part from interval algebra (Allen,
9292 1984). The TimeBank corpus provides TimeML annotations for 186 documents (Pustejovsky
9293 et al., 2003). Methods for detecting these temporal relations combine supervised machine
9294 learning with temporal constraints, such as transitivity (e.g. Mani et al., 2006; Chambers
9295 and Jurafsky, 2008).

9296 More recent annotation schemes and datasets combine temporal and causal relations (Mirza
9297 et al., 2014; Dunietz et al., 2017): for example, the CaTeRS dataset includes annotations
9298 of 320 five-sentence short stories (Mostafazadeh et al., 2016). Abstracting still further,
9299 processes are networks of causal relations between multiple events. A small dataset of
9300 biological processes is annotated in the ProcessBank dataset (Berant et al., 2014), with the
9301 goal of supporting automatic question answering on scientific textbooks.

	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.

9302 17.4 Hedges, denials, and hypotheticals

9303 The methods described thus far apply to propositions about the way things are in the
 9304 real world. But natural language can also describe events and relations that are likely or
 9305 unlikely, possible or impossible, desired or feared. The following examples hint at the scope
 9306 of the problem (Prabhakaran et al., 2010):

- 9307 (17.16) GM will lay off workers.
- 9308 (17.17) A spokesman for GM said GM will lay off workers.
- 9309 (17.18) GM may lay off workers.
- 9310 (17.19) The politician claimed that GM will lay off workers.
- 9311 (17.20) Some wish GM would lay off workers.
- 9312 (17.21) Will GM lay off workers?
- 9313 (17.22) Many wonder whether GM will lay off workers.

9314 Accurate information extraction requires handling these extra-propositional aspects
 9315 of meaning, which are sometimes summarized under the terms modality and negation.⁷
 9316 Modality refers to expressions of the speaker’s attitude towards her own statements, including
 9317 “degree of certainty, reliability, subjectivity, sources of information, and perspective” (Morante
 9318 and Sporleder, 2012). Various systematizations of modality have been proposed (e.g.,
 9319 Palmer, 2001), including categories such as future, interrogative, imperative, conditional,
 9320 and subjective. Information extraction is particularly concerned with negation and certainty.
 9321 For example, Saurí and Pustejovsky (2009) link negation with a modal calculus of certainty,

⁷The classification of negation as extra-propositional is controversial: Packard et al. (2014) argue that negation is a “core part of compositionally constructed logical-form representations.” Negation is an element of the semantic parsing tasks discussed in chapter 12 and chapter 13 — for example, negation markers are treated as adjuncts in PropBank semantic role labeling. However, many of the relation extraction methods mentioned in this chapter do not handle negation directly. A further consideration is that negation interacts closely with aspects of modality that are generally not considered in propositional semantics, such as certainty and subjectivity.

likelihood, and possibility, creating the two-dimensional schema shown in Table 17.4. This is the basis for the FactBank corpus, with annotations of the factuality of all sentences in 208 documents of news text.

A related concept is hedging, in which speakers limit their commitment to a proposition (Lakoff, 1973):

(17.23) These results suggest that expression of c-jun, jun B and jun D genes might be involved in terminal granulocyte differentiation... (Morante and Daelemans, 2009)

(17.24) A whale is technically a mammal (Lakoff, 1973)

In the first example, the hedges suggest and might communicate uncertainty; in the second example, there is no uncertainty, but the hedge technically indicates that the evidence for the proposition will not fully meet the reader's expectations. Hedging has been studied extensively in scientific texts (Medlock and Briscoe, 2007; Morante and Daelemans, 2009), where the goal of large-scale extraction of scientific facts is obstructed by hedges and speculation. Still another related aspect of modality is evidentiality, in which speakers mark the source of their information. In many languages, it is obligatory to mark evidentiality through affixes or particles (Aikhenvald, 2004); while evidentiality is not grammaticalized in English, authors are expected to express this information in contexts such as journalism (Kovach and Rosenstiel, 2014) and Wikipedia.⁸

Methods for handling negation and modality generally include two phases:

1. detecting negated or uncertain events;
2. identifying the scope and focus of the negation or modal operator.

A considerable body of work on negation has employed rule-based techniques such as regular expressions (Chapman et al., 2001) to detect negated events. Such techniques 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 are more fine grained, as shown in the example from Morante and Sporleder (2012):

(17.25) [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

⁸<https://en.wikipedia.org/wiki/Wikipedia:Verifiability>

9355 each occurs within a scope: after his habit he said and after mine I asked _____
 9356 questions. These tasks are typically formalized as sequence labeling problems, with each
 9357 word token labeled as beginning, inside, or outside of a cue, focus, or scope span (see § 8.3).
 9358 Conventional sequence labeling approaches can then be applied, using surface features as
 9359 well as syntax (Velldal et al., 2012) and semantic analysis (Packard et al., 2014). Labeled
 9360 datasets include the BioScope corpus of biomedical texts (Vincze et al., 2008) and a shared
 9361 task dataset of detective stories by Arthur Conan Doyle (Morante and Blanco, 2012).

9362 17.5 Question answering and machine reading

9363 The victory of the Watson question-answering system against three top human players on
 9364 the game show Jeopardy! was a landmark moment for natural language processing (Ferrucci
 9365 et al., 2010). Game show questions are usually answered by factoids: entity names and short
 9366 phrases.⁹ The task of factoid question answering is therefore closely related to information
 9367 extraction, with the additional problem of accurately parsing the question.

9368 17.5.1 Formal semantics

9369 Semantic parsing is an effective method for question-answering in restricted domains such as
 9370 questions about geography and airline reservations (Zettlemoyer and Collins, 2005), and has
 9371 also been applied in “open-domain” settings such as question answering on Freebase (Berant
 9372 et al., 2013) and biomedical research abstracts (Poon and Domingos, 2009). One approach
 9373 is to convert the question into a lambda calculus expression that returns a boolean value:
 9374 for example, the question who is the mayor of the capital of Georgia? would be converted
 9375 to,

$$\lambda x. \exists y \text{ capital}(\text{Georgia}, y) \wedge \text{mayor}(y, x). \quad [17.22]$$

9376 This lambda expression can then be used to query an existing knowledge base, returning
 9377 “true” for all entities that satisfy it.

9378 17.5.2 Machine reading

9379 Recent work has focused on answering questions about specific textual passages, similar
 9380 to the reading comprehension examinations for young students (Hirschman et al., 1999).
 9381 This task has come to be known as machine reading.

⁹The broader landscape of question answering includes “why” questions (Why did Ahab continue to pursue the white whale?), “how questions” (How did Queequeg die?), and requests for summaries (What was Ishmael’s attitude towards organized religion?). For more, see Hirschman and Gaizauskas (2001).

9382 17.5.2.1 Datasets

9383 The machine reading problem can be formulated in a number of different ways. The most
9384 important distinction is what form the answer should take.

- 9385 • Multiple-choice question answering, as in the MCTest dataset of stories (Richardson
9386 et al., 2013) and the New York Regents Science Exams (Clark, 2015). In MCTest,
9387 the answer is deducible from the text alone, while in the science exams, the system
9388 must make inferences using an existing model of the underlying scientific phenomena.
9389 Here is an example from MCTest:

9390 (17.26) James the turtle was always getting into trouble. Sometimes he'd reach into
9391 the freezer and empty out all the food ...

- 9392 Q: What is the name of the trouble making turtle?
9393 (a) Fries
9394 (b) Pudding
9395 (c) James
9396 (d) Jane

- 9397 • Cloze-style “fill in the blank” questions, as in the CNN/Daily Mail comprehension
9398 task (Hermann et al., 2015), the Children’s Book Test (Hill et al., 2016), and the
9399 Who-did-What dataset (Onishi et al., 2016). In these tasks, the system must guess
9400 which word or entity completes a sentence, based on reading a passage of text. Here
9401 is an example from Who-did-What:

9402 (17.27) Q: Tottenham manager Juande Ramos has hinted he will allow _____ to
9403 leave if the Bulgaria striker makes it clear he is unhappy. (Onishi et al.,
9404 2016)

9405 The query sentence may be selected either from the story itself, or from an external
9406 summary. In either case, datasets can be created automatically by processing large
9407 quantities existing documents. An additional constraint is that that missing element
9408 from the cloze must appear in the main passage of text: for example, in Who-did-What,
9409 the candidates include all entities mentioned in the main passage. In the CNN/Daily
9410 Mail dataset, each entity name is replaced by a unique identifier, e.g., entity37. This
9411 ensures that correct answers can only be obtained by accurately reading the text,
9412 and not from external knowledge about the entities.

- 9413 • Extractive question answering, in which the answer is drawn from the original text.
9414 In WikiQA, answers are sentences (Yang et al., 2015). In the Stanford Question
9415 Answering Dataset (SQuAD), answers are words or short phrases (Rajpurkar et al.,
9416 2016):

- 9417 (17.28) In metereology, precipitation is any product of the condensation of atmospheric
 9418 water vapor that falls under gravity.
 9419 Q: What causes precipitation to fall? A: gravity

9420 In both WikiQA and SQuAD, the original texts are Wikipedia articles, and the
 9421 questions are generated by crowdworkers.

9422 17.5.2.2 Methods

9423 A baseline method is to search the text for sentences or short passages that overlap with
 9424 both the query and the candidate answer (Richardson et al., 2013). In example (17.26), this
 9425 baseline would select the correct answer, since James appears in a sentence that includes
 9426 the query terms trouble and turtle.

This baseline can be implemented as a neural architecture, using an attention mechanism (see § 18.3.1), which scores the similarity of the query to each part of the source text (Chen et al., 2016). The first step is to encode the passage $\mathbf{w}^{(p)}$ and the query $\mathbf{w}^{(q)}$, using two bidirectional LSTMs (§ 7.6).

$$\mathbf{h}^{(q)} = \text{BiLSTM}(\mathbf{w}^{(q)}; \Theta^{(q)}) \quad [17.23]$$

$$\mathbf{h}^{(p)} = \text{BiLSTM}(\mathbf{w}^{(p)}; \Theta^{(p)}). \quad [17.24]$$

The query is represented by vertically concatenating the final states of the left-to-right and right-to-left passes:

$$\mathbf{u} = [\overrightarrow{\mathbf{h}}^{(q)}_{M_q}; \overleftarrow{\mathbf{h}}^{(q)}_0]. \quad [17.25]$$

The attention vector is computed as a softmax over a vector of bilinear products, and the expected representation is computed by summing over attention values,

$$\tilde{\alpha}_m = (\mathbf{u}^{(q)})^\top \mathbf{W}_a \mathbf{h}_m^{(p)} \quad [17.26]$$

$$\boldsymbol{\alpha} = \text{SoftMax}(\tilde{\boldsymbol{\alpha}}) \quad [17.27]$$

$$\mathbf{o} = \sum_{m=1}^M \alpha_m \mathbf{h}_m^{(p)}. \quad [17.28]$$

Each candidate answer c is represented by a vector \mathbf{x}_c . Assuming the candidate answers are spans from the original text, these vectors can be set equal to the corresponding element in $\mathbf{h}^{(p)}$. The score for each candidate answer a is computed by the inner product,

$$\hat{c} = \underset{c}{\operatorname{argmax}} \mathbf{o} \cdot \mathbf{x}_c. \quad [17.29]$$

9427 This architecture can be trained end-to-end from a loss based on the log-likelihood of
9428 the correct answer. A number of related architectures have been proposed (e.g., Hermann
9429 et al., 2015; Kadlec et al., 2016; Dhingra et al., 2017; Cui et al., 2017), and the relationships
9430 between these methods are surveyed by Wang et al. (2017).

9431 Additional resources

9432 The field of information extraction is surveyed in course notes by Grishman (2012), and
9433 more recently in a short survey paper (Grishman, 2015). Shen et al. (2015) survey the task
9434 of entity linking, and Ji and Grishman (2011) survey work on knowledge base population.
9435 This chapter’s discussion of non-propositional meaning was strongly influenced by Morante
9436 and Sporleder (2012), who introduced a special issue of the journal Computational Linguistics
9437 dedicated to recent work on modality and negation.

9438 Exercises

9439 1. Consider the following heuristic for entity linking:

- 9440 • Among all entities that have the same type as the mention (e.g., LOC, PER),
9441 choose the one whose name has the lowest edit distance from the mention.
9442 • If more than one entity has the right type and the lowest edit distance from the
9443 mention, choose the most popular one.
9444 • If no candidate entity has the right type, choose nil.

Now suppose you have the following feature function:

$$\mathbf{f}(y, \mathbf{x}) = [\text{edit-dist}(\text{name}(y), \mathbf{x}), \text{same-type}(y, \mathbf{x}), \text{popularity}(y), \delta(y = \text{nil})]$$

9445 Design a set of ranking weights θ that match the heuristic. You may assume that
9446 edit distance and popularity are always in the range [0, 100], and that the nil entity
9447 has values of zero for all features except $\delta(y = \text{nil})$.

9448 2. Now consider another heuristic:

- 9449 • Among all candidate entities that have edit distance zero from the mention and
9450 the right type, choose the most popular one.
9451 • If no entity has edit distance zero from the mention, choose the one with the
9452 right type that is most popular, regardless of edit distance.
9453 • If no entity has the right type, choose nil.

Using the same features and assumptions from the previous problem, prove that there is no set of weights that could implement this heuristic. Then show that the heuristic can be implemented by adding a single feature. Your new feature should consider only the edit distance.

3. * Consider the following formulation for collective entity linking, which rewards sets of entities that are all of the same type, where “types” can be elements of any set:

$$\psi_c(\mathbf{y}) = \begin{cases} \alpha & \text{all entities in } \mathbf{y} \text{ have the same type} \\ \beta & \text{more than half of the entities in } \mathbf{y} \text{ have the same type} \\ 0 & \text{otherwise.} \end{cases} \quad [17.30]$$

Show how to implement this model of collective entity linking in an integer linear program. You may want to review § 13.2.2.

To get started, here is an integer linear program for entity linking, without including the collective term ψ_c :

$$\begin{aligned} \max_{z_{i,y} \in \{0,1\}} \quad & \sum_{i=1}^N \sum_{y \in \mathcal{Y}(\mathbf{x}^{(i)})} s_{i,y} z_{i,y} \\ \text{s.t.} \quad & \sum_{y \in \mathcal{Y}(\mathbf{x}^{(i)})} z_{i,y} \leq 1 \quad \forall i \in \{1, 2, \dots, N\} \end{aligned}$$

where $z_{i,y} = 1$ if entity y is linked to mention i , and $s_{i,y}$ is a parameter that scores the quality of this individual ranking decision, e.g., $s_{i,y} = \boldsymbol{\theta} \cdot \mathbf{f}(y, \mathbf{x}^{(i)}, \mathbf{c}^{(i)})$.

To incorporate the collective linking score, you may assume parameters r ,

$$r_{y,\tau} = \begin{cases} 1, & \text{entity } y \text{ has type } \tau \\ 0, & \text{otherwise.} \end{cases} \quad [17.31]$$

Hint: You will need to define several auxiliary variables to optimize over.

4. Run `nltk.corpus.download('reuters')` to download the Reuters corpus in NLTK, and run from `nltk.corpus import reuters` to import it. The command `reuters.words()` returns an iterator over the tokens in the corpus.

- a) Apply the pattern _____, such as _____ to this corpus, obtaining candidates for the is-a relation, e.g. `is-a(Romania, Country)`. What are three pairs that this method identifies correctly? What are three different pairs that it gets wrong?
- b) Design a pattern for the president relation, e.g. `president(Philippines, Corazon Aquino)`. In this case, you may want to augment your pattern matcher with the ability to match multiple token wildcards, perhaps using case information to detect proper names. Again, list three correct

- 9476 c) Preprocess the Reuters data by running a named entity recognizer, replacing
 9477 tokens with named entity spans when applicable. Apply your president matcher
 9478 to this new data. Does the accuracy improve? Compare 20 randomly-selected
 9479 pairs from this pattern and the one you designed in the previous part.
- 9480 5. Represent the dependency path $\mathbf{x}^{(i)}$ as a sequence of words and dependency arcs
 9481 of length M_i , ignoring the endpoints of the path. In example 1 of Table 17.2, the
 9482 dependency path is,

$$\mathbf{x}^{(1)} = (\xleftarrow[\text{Nsubj}]{\text{traveled}} \xrightarrow[\text{Obl}]{}) \quad [17.32]$$

9483 If $x_m^{(i)}$ is a word, then let $\text{pos}(x_m^{(i)})$ be its part-of-speech, using the tagset defined in
 9484 chapter 8.

We can define the following kernel function over pairs of dependency paths (Bunescu and Mooney, 2005):

$$\kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \begin{cases} 0, & M_i \neq M_j \\ \prod_{m=1}^{M_i} c(x_m^{(i)}, x_m^{(j)}), & M_i = M_j \end{cases}$$

$$c(x_m^{(i)}, x_m^{(j)}) = \begin{cases} 2, & x_m^{(i)} = x_m^{(j)} \\ 1, & x_m^{(i)} \text{ and } x_m^{(j)} \text{ are words and } \text{pos}(x_m^{(i)}) = \text{pos}(x_m^{(j)}) \\ 0, & \text{otherwise.} \end{cases}$$

9485 Using this kernel function, compute the kernel similarities of example 1 from Table 17.2
 9486 with the other five examples.

6. Continuing from the previous problem, suppose that the instances have the following labels:

$$y_2 = 1, y_3 = -1, y_4 = -1, y_5 = 1, y_6 = 1 \quad [17.33]$$

9487 Identify the conditions for α and b under which $\hat{y}_1 = 1$. Remember the constraint
 9488 that $\alpha_i \geq 0$ for all i .

9489 Chapter 18

9490 Machine translation

9491 Machine translation (MT) is one of the “holy grail” problems in artificial intelligence, with
9492 the potential to transform society by facilitating communication between people anywhere
9493 in the world. As a result, MT has received significant attention and funding since the
9494 early 1950s. However, it has proved remarkably challenging, and while there has been
9495 substantial progress towards usable MT systems — especially for high-resource language
9496 pairs like English-French — we are still far from translation systems that match the nuance
9497 and depth of human translations.

9498 18.1 Machine translation as a task

9499 Machine translation can be formulated as an optimization problem:

$$\hat{\mathbf{w}}^{(t)} = \underset{\mathbf{w}^{(t)}}{\operatorname{argmax}} \Psi(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}), \quad [18.1]$$

9500 where $\mathbf{w}^{(s)}$ is a sentence in a source language, $\mathbf{w}^{(t)}$ is a sentence in the target language,
9501 and Ψ is a scoring function. As usual, this formalism requires two components: a decoding
9502 algorithm for computing $\hat{\mathbf{w}}^{(t)}$, and a learning algorithm for estimating the parameters of
9503 the scoring function Ψ .

9504 Decoding is difficult for machine translation because of the huge space of possible
9505 translations. We have faced large label spaces before: for example, in sequence labeling,
9506 the set of possible label sequences is exponential in the length of the input. In these
9507 cases, it was possible to search the space quickly by introducing locality assumptions: for
9508 example, that each tag depends only on its predecessor, or that each production depends
9509 only on its parent. In machine translation, no such locality assumptions seem possible:
9510 human translators reword, reorder, and rearrange words; they replace single words with
9511 multi-word phrases, and vice versa. This flexibility means that in even relatively simple

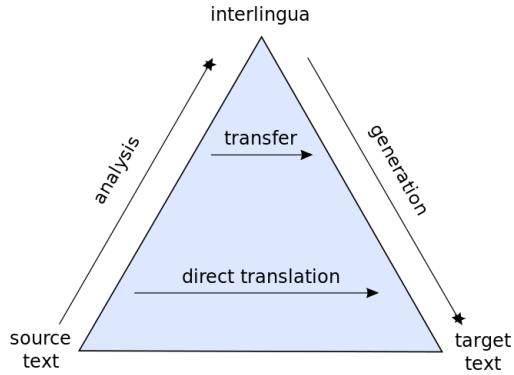


Figure 18.1: The Vauquois Pyramid http://commons.wikimedia.org/wiki/File:Direct_translation_and_transfer_translation_pyramind.svg

9512 translation models, decoding is NP-hard (Knight, 1999). Approaches for dealing with this
 9513 complexity are described in § 18.4.

Estimating translation models is difficult as well. Labeled translation data usually comes in the form parallel sentences, e.g.,

$$\begin{aligned} \mathbf{w}^{(s)} &= \text{A Vinay le gusta las manzanas.} \\ \mathbf{w}^{(t)} &= \text{Vinay likes apples.} \end{aligned}$$

9514 A useful feature function would note the translation pairs (gusta, likes), (manzanas, apples),
 9515 and even (Vinay, Vinay). But this word-to-word alignment is not given in the data. One
 9516 solution is to treat this alignment as a latent variable; this is the approach taken by
 9517 classical statistical machine translation (SMT) systems, described in § 18.2. Another
 9518 solution is to model the relationship between $\mathbf{w}^{(t)}$ and $\mathbf{w}^{(s)}$ through a more complex
 9519 and expressive function; this is the approach taken by neural machine translation (NMT)
 9520 systems, described in § 18.3.

9521 The Vauquois Pyramid is a theory of how translation should be done. At the lowest
 9522 level, the translation system operates on individual words, but the horizontal distance at
 9523 this level is large, because languages express ideas differently. If we can move up the triangle
 9524 to syntactic structure, the distance for translation is reduced; we then need only produce
 9525 target-language text from the syntactic representation, which can be as simple as reading off
 9526 a tree. Further up the triangle lies semantics; translating between semantic representations
 9527 should be easier still, but mapping between semantics and surface text is a difficult,
 9528 unsolved problem. At the top of the triangle is interlingua, a semantic representation that
 9529 is so generic that it is identical across all human languages. Philosophers debate whether

	Adequate?	Fluent?
To Vinay it like Python	yes	no
Vinay debugs memory leaks	no	yes
Vinay likes Python	yes	yes

Table 18.1: Adequacy and fluency for translations of the Spanish sentence A Vinay le gusta Python.

such a thing as interlingua is really possible (Derrida, 1985). While the first-order logic representations discussed in chapter 12 might be considered to be language independent, it is built on an inventory of relations that is suspiciously similar to a subset of English words (Nirenburg and Wilks, 2001). Nonetheless, the idea of linking translation and semantic understanding may still be a promising path, if the resulting translations better preserve the meaning of the original text.

18.1.1 Evaluating translations

There are two main criteria for a translation, summarized in Table 18.1.

- Adequacy: The translation $\mathbf{w}^{(t)}$ should adequately reflect the linguistic content of $\mathbf{w}^{(s)}$. For example, if $\mathbf{w}^{(s)} = \text{A Vinay le gusta Python}$, the gloss¹ $\mathbf{w}^{(t)} = \text{To Vinay it like Python}$ is considered adequate because it contains all the relevant content. The output $\mathbf{w}^{(t)} = \text{Vinay debugs memory leaks}$ is not adequate.
- Fluency: The translation $\mathbf{w}^{(t)}$ should read like fluent text in the target language. By this criterion, the gloss $\mathbf{w}^{(t)} = \text{To Vinay it like Python}$ will score poorly, and $\mathbf{w}^{(t)} = \text{Vinay debugs memory leaks}$ will be preferred.

Automated evaluations of machine translations typically merge both of these criteria, by comparing the system translation with one or more reference translations, produced by professional human translators. The most popular quantitative metric is bleu (bilingual evaluation underway; Papineni et al., 2002), which is based on n -gram precision: what fraction of n -grams in the system translation appear in the reference? Specifically, for each n -gram length, the precision is defined as,

$$p_n = \frac{\text{number of } n\text{-grams appearing in both reference and hypothesis translations}}{\text{number of } n\text{-grams appearing in the hypothesis translation}}. \quad [18.2]$$

The n -gram precisions for three hypothesis translations are shown in Figure 18.2.

¹A gloss is a word-for-word translation.

	Translation	p_1	p_2	p_3	p_4	BP	BLEU
Reference	Vinay likes programming in Python						
Sys1	To Vinay it like to program Python	$\frac{2}{7}$	0	0	0	1	.21
Sys2	Vinay likes Python	$\frac{3}{3}$	$\frac{1}{2}$	0	0	.51	.33
Sys3	Vinay likes programming in his pajamas	$\frac{4}{6}$	$\frac{3}{5}$	$\frac{2}{4}$	$\frac{1}{3}$	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.

9552 The bleu score is then based on the average, $\exp \frac{1}{N} \sum_{n=1}^N \log p_n$. Two modifications of
 9553 Equation 18.2 are necessary: (1) to avoid computing $\log 0$, all precisions are smoothed to
 9554 ensure that they are positive; (2) each n -gram in the source can be used at most once,
 9555 so that to to to to to does not achieve $p_1 = 1$ against the reference to be or not to
 9556 be. Furthermore, precision-based metrics are biased in favor of short translations, which
 9557 can achieve high scores by minimizing the denominator in [18.2]. To avoid this issue, a
 9558 brevity penalty is applied to translations that are shorter than the reference. This penalty
 9559 is indicated as “BP” in Figure 18.2.

9560 Automated metrics like bleu have been validated by correlation with human judgments
 9561 of translation quality. Nonetheless, it is not difficult to construct examples in which the
 9562 bleu score is high, yet the translation is disfluent or carries a completely different meaning
 9563 from the original. To give just one example, consider the problem of translating pronouns.
 9564 Because pronouns refer to specific entities, a single incorrect pronoun can obliterate the
 9565 semantics of the original sentence. Existing state-of-the-art systems generally do not
 9566 attempt the reasoning necessary to correctly resolve pronominal anaphora (Hardmeier,
 9567 2012). Despite the importance of pronouns for semantics, they have a marginal impact
 9568 on bleu, which may help to explain why existing systems do not make a greater effort to
 9569 translate them correctly.

9570 Fairness and bias The problem of pronoun translation intersects with issues of fairness
 9571 and bias. In many languages, such as Turkish, the third person singular pronoun is
 9572 gender neutral. Today’s state-of-the-art systems produce the following Turkish-English
 9573 translations (Caliskan et al., 2017):

- 9574 (18.1) O bir doktor.
 He is a doctor.
 9575 (18.2) O bir hemşire.
 She is a nurse.

9576 The same problem arises for other professions that have stereotypical genders, such as
9577 engineers, soldiers, and teachers, and for other languages that have gender-neutral pronouns.
9578 This bias was not directly programmed into the translation model; it arises from statistical
9579 tendencies in existing datasets. This highlights a general problem with data-driven approaches,
9580 which can perpetuate biases that negatively impact disadvantaged groups. Worse, machine
9581 learning can amplify biases in data (Bolukbasi et al., 2016): if a dataset has even a slight
9582 tendency towards men as doctors, the resulting translation model may produce translations
9583 in which doctors are always he, and nurses are always she.

9584 Other metrics A range of other automated metrics have been proposed for machine
9585 translation. One potential weakness of bleu is that it only measures precision; meteor
9586 is a weighted F -measure, which is a combination of recall and precision (see § 4.4.1).
9587 Translation Error Rate (ter) computes the string edit distance (see § 9.1.4.1) between the
9588 reference and the hypothesis (Snover et al., 2006). For language pairs like English and
9589 Japanese, there are substantial differences in word order, and word order errors are not
9590 sufficiently captured by n -gram based metrics. The ribes metric applies rank correlation to
9591 measure the similarity in word order between the system and reference translations (Isozaki
9592 et al., 2010).

9593 18.1.2 Data

9594 Data-driven approaches to machine translation rely primarily on parallel corpora: sentence-level
9595 translations. Early work focused on government records, in which fine-grained official
9596 translations are often required. For example, the IBM translation systems were based
9597 on the proceedings of the Canadian Parliament, called Hansards, which are recorded in
9598 English and French (Brown et al., 1990). The growth of the European Union led to the
9599 development of the EuroParl corpus, which spans 21 European languages (Koehn, 2005).
9600 While these datasets helped to launch the field of machine translation, they are restricted
9601 to narrow domains and a formal speaking style, limiting their applicability to other types
9602 of text. As more resources are committed to machine translation, new translation datasets
9603 have been commissioned. This has broadened the scope of available data to news,² movie
9604 subtitles,³ social media (Ling et al., 2013), dialogues (Fordyce, 2007), TED talks (Paul
9605 et al., 2010), and scientific research articles (Nakazawa et al., 2016).

9606 Despite this growing set of resources, the main bottleneck in machine translation data
9607 is the need for parallel corpora that are aligned at the sentence level. Many languages have
9608 sizable parallel corpora with some high-resource language, but not with each other. The
9609 high-resource language can then be used as a “pivot” or “bridge” (Boitet, 1988; Utiyama
9610 and Isahara, 2007): for example, De Gispert and Marino (2006) use Spanish as a bridge for

²<https://catalog.ldc.upenn.edu/LDC2010T10>, <http://www.statmt.org/wmt15/translation-task.html>

³<http://opus.nlpl.eu/>

translation between Catalan and English. For most of the 6000 languages spoken today, the only source of translation data remains the Judeo-Christian Bible (Resnik et al., 1999). While relatively small, at less than a million tokens, the Bible has been translated into more than 2000 languages, far outpacing any other corpus. Some research has explored the possibility of automatically identifying parallel sentence pairs from unaligned parallel texts, such as web pages and Wikipedia articles (Kilgarriff and Grefenstette, 2003; Resnik and Smith, 2003; Adafre and De Rijke, 2006). Another approach is to create large parallel corpora through crowdsourcing (Zaidan and Callison-Burch, 2011).

18.2 Statistical machine translation

The previous section introduced adequacy and fluency as the two main criteria for machine translation. A natural modeling approach is to represent them with separate scores,

$$\Psi(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \Psi_A(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) + \Psi_F(\mathbf{w}^{(t)}). \quad [18.3]$$

The fluency score Ψ_F need not even consider the source sentence; it only judges $\mathbf{w}^{(t)}$ on whether it is fluent in the target language. This decomposition is advantageous because it makes it possible to estimate the two scoring functions on separate data. While the adequacy model must be estimated from aligned sentences — which are relatively expensive and rare — the fluency model can be estimated from monolingual text in the target language. Large monolingual corpora are now available in many languages, thanks to resources such as Wikipedia.

An elegant justification of the decomposition in Equation 18.3 is provided by the noisy channel model, in which each scoring function is a log probability:

$$\Psi_A(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) \triangleq \log p_{S|T}(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}) \quad [18.4]$$

$$\Psi_F(\mathbf{w}^{(t)}) \triangleq \log p_T(\mathbf{w}^{(t)}) \quad [18.5]$$

$$\Psi(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \log p_{S|T}(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}) + \log p_T(\mathbf{w}^{(t)}) = \log p_{S,T}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}). \quad [18.6]$$

By setting the scoring functions equal to the logarithms of the prior and likelihood, their sum is equal to $\log p_{S,T}$, which is the logarithm of the joint probability of the source and target. The sentence $\hat{\mathbf{w}}^{(t)}$ that maximizes this joint probability is also the maximizer of the conditional probability $p_{T|S}$, making it the most likely target language sentence, conditioned on the source.

The noisy channel model can be justified by a generative story. The target text is originally generated from a probability model p_T . It is then encoded in a “noisy channel” $p_{S|T}$, which converts it to a string in the source language. In decoding, we apply Bayes’ rule to recover the string $\mathbf{w}^{(t)}$ that is maximally likely under the conditional probability $p_{T|S}$. Under this interpretation, the target probability p_T is just a language model, and

	A	Vinay	le	gusta	python
Vinay					
likes					
python					

A Vinay le gusta python

Figure 18.3: An example word-to-word alignment

9639 can be estimated using any of the techniques from chapter 6. The only remaining learning
 9640 problem is to estimate the translation model $p_{S|T}$.

9641 18.2.1 Statistical translation modeling

9642 The simplest decomposition of the translation model is word-to-word: each word in the
 9643 source should be aligned to a word in the translation. This approach presupposes an
 9644 alignment $\mathcal{A}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)})$, which contains a list of pairs of source and target tokens. For
 9645 example, given $\mathbf{w}^{(s)} = \text{A Vinay le gusta Python}$ and $\mathbf{w}^{(t)} = \text{Vinay likes Python}$, one
 9646 possible word-to-word alignment is,

$$\mathcal{A}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \{(A, \emptyset), (\text{Vinay}, \text{Vinay}), (\text{le}, \text{likes}), (\text{gusta}, \text{likes}), (\text{Python}, \text{Python})\}. \quad [18.7]$$

9647 This alignment is shown in Figure 18.3. Another, less promising, alignment is:

$$\mathcal{A}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \{(A, \text{Vinay}), (\text{Vinay}, \text{likes}), (\text{le}, \text{Python}), (\text{gusta}, \emptyset), (\text{Python}, \emptyset)\}. \quad [18.8]$$

9648 Each alignment contains exactly one tuple for each word in the source, which serves to
 9649 explain how the source word could be translated from the target, as required by the
 9650 translation probability $p_{S|T}$. If no appropriate word in the target can be identified for
 9651 a source word, it is aligned to \emptyset — as is the case for the Spanish function word *a* in the
 9652 example, which glosses to the English word *to*. Words in the target can align with multiple
 9653 words in the source, so that the target word *likes* can align to both *le* and *gusta* in the
 9654 source.

The joint probability of the alignment and the translation can be defined conveniently

as,

$$p(\mathbf{w}^{(s)}, \mathcal{A} | \mathbf{w}^{(t)}) = \prod_{m=1}^{M^{(s)}} p(w_m^{(s)}, a_m | w_{a_m}^{(t)}, m, M^{(s)}, M^{(t)}) \quad [18.9]$$

$$= \prod_{m=1}^{M^{(s)}} p(a_m | m, M^{(s)}, M^{(t)}) \times p(w_m^{(s)} | w_{a_m}^{(t)}). \quad [18.10]$$

9655 This probability model makes two key assumptions:

- 9656 • The alignment probability factors across tokens,

$$p(\mathcal{A} | \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \prod_{m=1}^{M^{(s)}} p(a_m | m, M^{(s)}, M^{(t)}). \quad [18.11]$$

9657 This means that each alignment decision is independent of the others, and depends
9658 only on the index m , and the sentence lengths $M^{(s)}$ and $M^{(t)}$.

- 9659 • The translation probability also factors across tokens,

$$p(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}, \mathcal{A}) = \prod_{m=1}^{M^{(s)}} p(w_m^{(s)} | w_{a_m}^{(t)}), \quad [18.12]$$

9660 so that each word in $\mathbf{w}^{(s)}$ depends only on its aligned word in $\mathbf{w}^{(t)}$. This means that
9661 translation is word-to-word, ignoring context. The hope is that the target language
9662 model $p(\mathbf{w}^{(t)})$ will correct any disfluencies that arise from word-to-word translation.

To translate with such a model, we could sum or max over all possible alignments,

$$p(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \sum_{\mathcal{A}} p(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}, \mathcal{A}) \quad [18.13]$$

$$= p(\mathbf{w}^{(t)}) \sum_{\mathcal{A}} p(\mathcal{A}) \times p(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}, \mathcal{A}) \quad [18.14]$$

$$\geq p(\mathbf{w}^{(t)}) \max_{\mathcal{A}} p(\mathcal{A}) \times p(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}, \mathcal{A}). \quad [18.15]$$

The term $p(\mathcal{A})$ defines the prior probability over alignments. A series of alignment models with increasingly relaxed independence assumptions was developed by researchers at IBM in the 1980s and 1990s, known as IBM Models 1-6 (Och and Ney, 2003). IBM Model 1 makes the strongest independence assumption:

$$p(a_m | m, M^{(s)}, M^{(t)}) = \frac{1}{M^{(t)}}. \quad [18.16]$$

9663 In this model, every alignment is equally likely. This is almost surely wrong, but it results
9664 in a convex learning objective, yielding a good initialization for the more complex alignment
9665 models (Brown et al., 1993; Koehn, 2009).

9666 18.2.2 Estimation

9667 Let us define the parameter $\theta_{u \rightarrow v}$ as the probability of translating target word u to source
 9668 word v . If word-to-word alignments were annotated, these probabilities could be computed
 9669 from relative frequencies,

$$\hat{\theta}_{u \rightarrow v} = \frac{\text{count}(u, v)}{\text{count}(u)}, \quad [18.17]$$

9670 where $\text{count}(u, v)$ is the count of instances in which word v was aligned to word u in the
 9671 training set, and $\text{count}(u)$ is the total count of the target word u . The smoothing techniques
 9672 mentioned in chapter 6 can help to reduce the variance of these probability estimates.

9673 Conversely, if we had an accurate translation model, we could estimate the likelihood
 9674 of each alignment decision,

$$q_m(a_m | \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) \propto p(a_m | m, M^{(s)}, M^{(t)}) \times p(w_m^{(s)} | w_{a_m}^{(t)}), \quad [18.18]$$

where $q_m(a_m | \mathbf{w}^{(s)}, \mathbf{w}^{(t)})$ is a measure of our confidence in aligning source word $w_m^{(s)}$ to
 target word $w_{a_m}^{(t)}$. The relative frequencies could then be computed from the expected
 counts,

$$\hat{\theta}_{u \rightarrow v} = \frac{E_q[\text{count}(u, v)]}{\text{count}(u)} \quad [18.19]$$

$$E_q[\text{count}(u, v)] = \sum_m q_m(a_m | \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) \delta(w_m^{(s)} = v) \delta(w_{a_m}^{(t)} = u). \quad [18.20]$$

9675 The expectation-maximization (EM) algorithm proceeds by iteratively updating q_m
 9676 and $\hat{\Theta}$. The algorithm is described in general form in chapter 5. For statistical machine
 9677 translation, the steps of the algorithm are:

- 9678 1. E-step: Update beliefs about word alignment using Equation 18.18.
 9679 2. M-step: Update the translation model using Equations 18.19 and 18.20.

9680 As discussed in chapter 5, the expectation maximization algorithm is guaranteed to converge,
 9681 but not to a global optimum. However, for IBM Model 1, it can be shown that EM optimizes
 9682 a convex objective, and global optimality is guaranteed. For this reason, IBM Model 1 is
 9683 often used as an initialization for more complex alignment models. For more detail, see
 9684 Koehn (2009).

9685 18.2.3 Phrase-based translation

9686 Real translations are not word-to-word substitutions. One reason is that many multiword
 9687 expressions are not translated literally, as shown in this example from French:

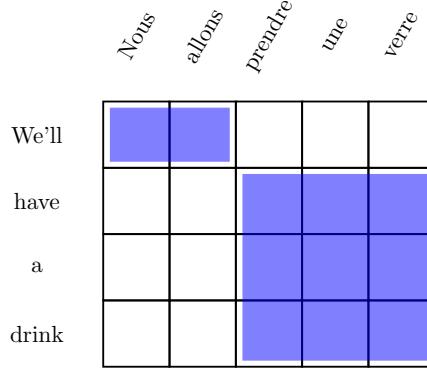


Figure 18.4: A phrase-based alignment between French and English, corresponding to example (18.3)

9688 (18.3) Nous allons prendre un verre
 We will take a glass
 9689 We'll have a drink

9690 The line we will take a glass is the word-for-word gloss of the French sentence; the
 9691 translation we'll have a drink is shown on the third line. Such examples are difficult
 9692 for word-to-word translation models, since they require translating prendre to have and
 9693 verre to drink. These translations are only correct in the context of these specific phrases.

Phrase-based translation generalizes on word-based models by building translation tables and alignments between multiword spans. (These “phrases” are not necessarily syntactic constituents like the noun phrases and verb phrases described in chapters 9 and 10.) The generalization from word-based translation is surprisingly straightforward: the translation tables can now condition on multi-word units, and can assign probabilities to multi-word units; alignments are mappings from spans to spans, $((i, j), (k, \ell))$, so that

$$p(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}, \mathcal{A}) = \prod_{((i,j),(k,\ell)) \in \mathcal{A}} p_{w^{(s)}|w^{(t)}}(\{w_{i+1}^{(s)}, w_{i+2}^{(s)}, \dots, w_j^{(s)}\} | \{w_{k+1}^{(t)}, w_{k+2}^{(t)}, \dots, w_\ell^{(t)}\}). \quad [18.21]$$

9694 The phrase alignment $((i, j), (k, \ell))$ indicates that the span $\mathbf{w}_{i+1:j}^{(s)}$ is the translation of
 9695 the span $\mathbf{w}_{k+1:\ell}^{(t)}$. An example phrasal alignment is shown in Figure 18.4. Note that the
 9696 alignment set \mathcal{A} is required to cover all of the tokens in the source, just as in word-based
 9697 translation. The probability model $p_{w^{(s)}|w^{(t)}}$ must now include translations for all phrase
 9698 pairs, which can be learned from expectation-maximization just as in word-based statistical
 9699 machine translation.

9700 18.2.4 *Syntax-based translation

9701 The Vauquois Pyramid (Figure 18.1) suggests that translation might be easier if we take
 9702 a higher-level view. One possibility is to incorporate the syntactic structure of the source,
 9703 the target, or both. This is particularly promising for language pairs that consistent
 9704 syntactic differences. For example, English adjectives almost always precede the nouns
 9705 that they modify, while in Romance languages such as French and Spanish, the adjective
 9706 often follows the noun: thus, angry fish would translate to pez (fish) enojado (angry) in
 9707 Spanish. In word-to-word translation, these reorderings cause the alignment model to be
 9708 overly permissive. It is not that the order of any pair of English words can be reversed
 9709 when translating into Spanish, but only adjectives and nouns within a noun phrase. Similar
 9710 issues arise when translating between verb-final languages such as Japanese (in which verbs
 9711 usually follow the subject and object), verb-initial languages like Tagalog and classical
 9712 Arabic, and verb-medial languages such as English.

9713 An elegant solution is to link parsing and translation in a synchronous context-free
 9714 grammar (SCFG; Chiang, 2007).⁴ An SCFG is a set of productions of the form $X \rightarrow (\alpha, \beta, \sim)$,
 9715 where X is a non-terminal, α and β are sequences of terminals or non-terminals, and \sim
 9716 is a one-to-one alignment of items in α with items in β . To handle the English-Spanish
 9717 adjective-noun ordering, an SCFG would include productions such as,

$$\text{NP} \rightarrow (\text{Det}_1 \text{Nn}_2 \text{Jj}_3, \quad \text{Det}_1 \text{Jj}_3 \text{Nn}_2), \quad [18.22]$$

9718 with subscripts indicating the alignment between the Spanish (left) and English (right)
 9719 parts of the right-hand side. Terminal productions yield translation pairs,

$$\text{Jj} \rightarrow (\text{enojado}_1, \text{angry}_1). \quad [18.23]$$

9720 A synchronous derivation begins with the start symbol S , and derives a pair of sequences
 9721 of terminal symbols.

9722 Given an SCFG in which each production yields at most two symbols in each language
 9723 (Chomsky Normal Form; see § 9.2.1.2), a sentence can be parsed using only the CKY
 9724 algorithm (chapter 10). The resulting derivation also includes productions in the other
 9725 language, all the way down to the surface form. Therefore, SCFGs make translation very
 9726 similar to parsing. In a weighted SCFG, the log probability $\log p_{S|T}$ can be computed
 9727 from the sum of the log-probabilities of the productions. However, combining SCFGs with
 9728 a target language model is computationally expensive, necessitating approximate search
 9729 algorithms (Huang and Chiang, 2007).

9730 Synchronous context-free grammars are an example of tree-to-tree translation, because
 9731 they model the syntactic structure of both the target and source language. In string-to-tree

⁴Key earlier work includes syntax-driven transduction (Lewis II and Stearns, 1968) and stochastic inversion transduction grammars (Wu, 1997).

translation, string elements are translated into constituent tree fragments, which are then assembled into a translation (Yamada and Knight, 2001; Galley et al., 2004); in tree-to-string translation, the source side is parsed, and then transformed into a string on the target side (Liu et al., 2006). A key question for syntax-based translation is the extent to which we phrasal constituents align across translations (Fox, 2002), because this governs the extent to which we can rely on monolingual parsers and treebanks. For more on syntax-based machine translation, see the monograph by Williams et al. (2016).

18.3 Neural machine translation

Neural network models for machine translation are based on the encoder-decoder architecture (Cho et al., 2014). The encoder network converts the source language sentence into a vector or matrix representation; the decoder network then converts the encoding into a sentence in the target language.

$$\mathbf{z} = \text{Encode}(\mathbf{w}^{(s)}) \quad [18.24]$$

$$\mathbf{w}^{(t)} \mid \mathbf{w}^{(s)} \sim \text{Decode}(\mathbf{z}), \quad [18.25]$$

where the second line means that the function $\text{Decode}(\mathbf{z})$ defines the conditional probability $p(\mathbf{w}^{(t)} \mid \mathbf{w}^{(s)})$.

The decoder is typically a recurrent neural network, which generates the target language sentence one word at a time, while recurrently updating a hidden state. The encoder and decoder networks are trained end-to-end from parallel sentences. If the output layer of the decoder is a logistic function, then the entire architecture can be trained to maximize the conditional log-likelihood,

$$\log p(\mathbf{w}^{(t)} \mid \mathbf{w}^{(s)}) = \sum_{m=1}^{M^{(t)}} p(w_m^{(t)} \mid \mathbf{w}_{1:m-1}^{(t)}, \mathbf{z}) \quad [18.26]$$

$$p(w_m^{(t)} \mid \mathbf{w}_{1:m-1}^{(t)}, \mathbf{w}^{(s)}) \propto \exp\left(\boldsymbol{\beta}_{w_m^{(t)}} \cdot \mathbf{h}_{m-1}^{(t)}\right) \quad [18.27]$$

where the hidden state $\mathbf{h}_{m-1}^{(t)}$ is a recurrent function of the previously generated text $\mathbf{w}_{1:m-1}^{(t)}$ and the encoding \mathbf{z} . The second line is equivalent to writing,

$$w_m^{(t)} \mid \mathbf{w}_{1:m-1}^{(t)}, \mathbf{w}^{(s)} \sim \text{SoftMax}\left(\boldsymbol{\beta} \cdot \mathbf{h}_{m-1}^{(t)}\right), \quad [18.28]$$

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 language vocabulary.

The simplest encoder-decoder architecture is the sequence-to-sequence model (Sutskever et al., 2014). In this model, the encoder is set to the final hidden state of a long short-term

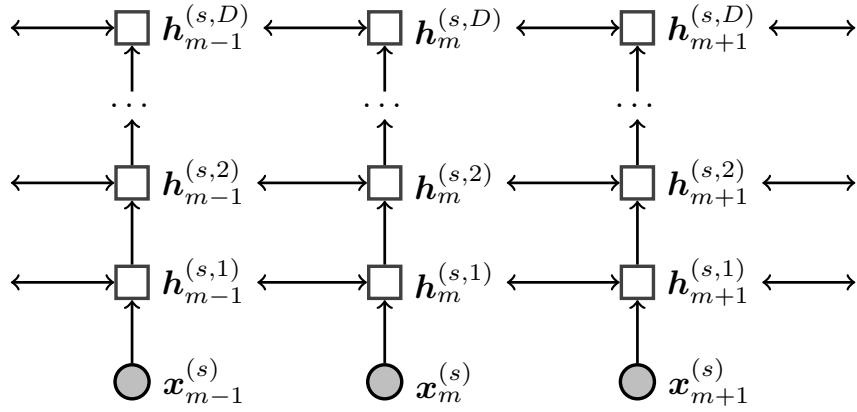


Figure 18.5: A deep bidirectional LSTM encoder

memory (LSTM) (see § 6.3.3) on the source sentence:

$$\mathbf{h}_m^{(s)} = \text{LSTM}(\mathbf{x}_m^{(s)}, \mathbf{h}_{m-1}^{(s)}) \quad [18.29]$$

$$\mathbf{z} \triangleq \mathbf{h}_{M^{(s)}}^{(s)}, \quad [18.30]$$

where $\mathbf{x}_m^{(s)}$ is the embedding of source language word $w_m^{(s)}$. The encoding then provides the initial hidden state for the decoder LSTM:

$$\mathbf{h}_0^{(t)} = \mathbf{z} \quad [18.31]$$

$$\mathbf{h}_m^{(t)} = \text{LSTM}(\mathbf{x}_m^{(t)}, \mathbf{h}_{m-1}^{(t)}), \quad [18.32]$$

9744 where $\mathbf{x}_m^{(t)}$ is the embedding of the target language word $w_m^{(t)}$.

9745 Sequence-to-sequence translation is nothing more than wiring together two LSTMs:
 9746 one to read the source, and another to generate the target. To make the model work well,
 9747 some additional tweaks are needed:

- 9748 • Most notably, the model works much better if the source sentence is reversed, reading
 9749 from the end of the sentence back to the beginning. In this way, the words at the
 9750 beginning of the source have the greatest impact on the encoding \mathbf{z} , and therefore
 9751 impact the words at the beginning of the target sentence. Later work on more
 9752 advanced encoding models, such as neural attention (see § 18.3.1), has eliminated
 9753 the need for reversing the source sentence.
- The encoder and decoder can be implemented as deep LSTMs, with multiple layers
 of hidden states. As shown in Figure 18.5, each hidden state $\mathbf{h}_m^{(s,i)}$ at layer i is treated

as the input to an LSTM at layer $i + 1$:

$$\mathbf{h}_m^{(s,1)} = \text{LSTM}(\mathbf{x}_m^{(s)}, \mathbf{h}_{m-1}^{(s)}) \quad [18.33]$$

$$\mathbf{h}_m^{(s,i+1)} = \text{LSTM}(\mathbf{h}_m^{(s,i)}, \mathbf{h}_{m-1}^{(s)}), \quad \forall i \geq 1. \quad [18.34]$$

9754 The original work on sequence-to-sequence translation used four layers; in 2016,
 9755 Google’s commercial machine translation system used eight layers (Wu et al., 2016).⁵

- 9756 • Significant improvements can be obtained by creating an ensemble of translation
 9757 models, each trained from a different random initialization. For an ensemble of size
 9758 N , the per-token decoding probability is set equal to,

$$p(w^{(t)} | \mathbf{z}, \mathbf{w}_{1:m-1}^{(t)}) = \frac{1}{N} \sum_{i=1}^N p_i(w^{(t)} | \mathbf{z}, \mathbf{w}_{1:m-1}^{(t)}), \quad [18.35]$$

9759 where p_i is the decoding probability for model i . Each translation model in the
 9760 ensemble includes its own encoder and decoder networks.

- 9761 • The original sequence-to-sequence model used a fairly standard training setup: stochastic
 9762 gradient descent with an exponentially decreasing learning rate after the first five
 9763 epochs; mini-batches of 128 sentences, chosen to have similar length so that each
 9764 sentence on the batch will take roughly the same amount of time to process; gradient
 9765 clipping (see § 3.3.4) to ensure that the norm of the gradient never exceeds some
 9766 predefined value.

9767 18.3.1 Neural attention

9768 The sequence-to-sequence model discussed in the previous section was a radical departure
 9769 from statistical machine translation, in which each word or phrase in the target language
 9770 is conditioned on a single word or phrase in the source language. Both approaches have
 9771 advantages. Statistical translation leverages the idea of compositionality — translations of
 9772 large units should be based on the translations of their component parts — and this seems
 9773 crucial if we are to scale translation to longer units of text. But the translation of each
 9774 word or phrase often depends on the larger context, and encoder-decoder models capture
 9775 this context at the sentence level.

9776 Is it possible for translation to be both contextualized and compositional? One approach
 9777 is to augment neural translation with an attention mechanism. The idea of neural attention
 9778 was described in § 17.5, but its application to translation bears further discussion. In
 9779 general, attention can be thought of as using a query to select from a memory of key-value
 9780 pairs. However, the query, keys, and values are all vectors, and the entire operation is

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.

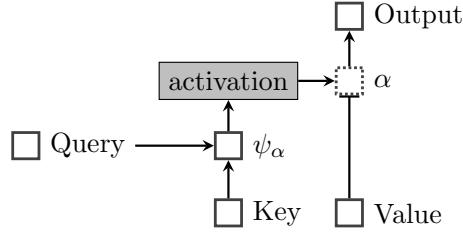


Figure 18.6: A general view of neural attention. The dotted box indicates that each $\alpha_{m \rightarrow n}$ can be viewed as a gate on value n .

differentiable. For each key n in the memory, we compute a score $\psi_\alpha(m, n)$ with respect to the query m . That score is a function of the compatibility of the key and the query, and can be computed using a small feedforward neural network. The vector of scores is passed through an activation function, such as softmax. The output of this activation function is a vector of non-negative numbers $[\alpha_{m \rightarrow 1}, \alpha_{m \rightarrow 2}, \dots, \alpha_{m \rightarrow N}]^\top$, with length N equal to the size of the memory. Each value in the memory v_n is multiplied by the attention $\alpha_{m \rightarrow n}$; the sum of these scaled values is the output. This process is shown in Figure 18.6. In the extreme case that $\alpha_{m \rightarrow n} = 1$ and $\alpha_{m \rightarrow n'} = 0$ for all other n' , then the attention mechanism simply selects the value v_n from the memory.

Neural attention makes it possible to integrate alignment into the encoder-decoder architecture. Rather than encoding the entire source sentence into a fixed length vector \mathbf{z} , it can be encoded into a matrix $\mathbf{Z} \in \mathbb{R}^{K \times M^{(S)}}$, where K is the dimension of the hidden state, and $M^{(S)}$ is the number of tokens in the source input. Each column of \mathbf{Z} represents the state of a recurrent neural network over the source sentence. These vectors are constructed from a bidirectional LSTM (see § 7.6), which can be a deep network as shown in Figure 18.5. These columns are both the keys and the values in the attention mechanism.

At each step m in decoding, the attentional state is computed by executing a query, which is equal to the state of the decoder, $\mathbf{h}_m^{(t)}$. The resulting compatibility scores are,

$$\psi_\alpha(m, n) = \mathbf{v}_\alpha \cdot \tanh(\Theta_\alpha[\mathbf{h}_m^{(t)}; \mathbf{h}_n^{(s)}]). \quad [18.36]$$

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:

Softmax attention

$$\alpha_{m \rightarrow n} = \frac{\exp \psi_\alpha(m, n)}{\sum_{n'=1}^{M^{(s)}} \exp \psi_\alpha(m, n')} \quad [18.37]$$

Sigmoid attention

$$\alpha_{m \rightarrow n} = \sigma(\psi_\alpha(m, n)) \quad [18.38]$$

The attention α is then used to compute an context vector \mathbf{c}_m by taking a weighted average over the columns of \mathbf{Z} ,

$$\mathbf{c}_m = \sum_{n=1}^{M^{(s)}} \alpha_{m \rightarrow n} \mathbf{z}_n, \quad [18.39]$$

where $\alpha_{m \rightarrow n} \in [0, 1]$ is the amount of attention from word m of the target to word n of the source. The context vector can be incorporated into the decoder’s word output probability model, by adding another layer to the decoder (Luong et al., 2015):

$$\tilde{\mathbf{h}}_m^{(t)} = \tanh(\Theta_c[\mathbf{h}_m^{(t)}; \mathbf{c}_m]) \quad [18.40]$$

$$p(w_{m+1}^{(t)} | \mathbf{w}_{1:m}^{(t)}, \mathbf{w}^{(s)}) \propto \exp\left(\beta_{w_{m+1}^{(t)}} \cdot \tilde{\mathbf{h}}_m^{(t)}\right). \quad [18.41]$$

9801 Here the decoder state $\mathbf{h}_m^{(t)}$ is concatenated with the context vector, forming the input
 9802 to compute a final output vector $\tilde{\mathbf{h}}_m^{(t)}$. The context vector can be incorporated into the
 9803 decoder recurrence in a similar manner (Bahdanau et al., 2014).

9804 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:

$$\mathbf{z}_m^{(i)} = \sum_{n=1}^{M^{(s)}} \alpha_{m \rightarrow n}^{(i)} (\Theta_v \mathbf{h}_n^{(i-1)}) \quad [18.42]$$

$$\mathbf{h}_m^{(i)} = \Theta_2 \text{ReLU}\left(\Theta_1 \mathbf{z}_m^{(i)} + \mathbf{b}_1\right) + \mathbf{b}_2. \quad [18.43]$$

9805 For each token m at level i , we compute self-attention over the entire source sentence:
 9806 the keys, values, and queries are all projections of the vector $\mathbf{h}^{(i-1)}$. The attention scores
 9807 $\alpha_{m \rightarrow n}^{(i)}$ are computed using a scaled form of softmax attention,

$$\alpha_{m \rightarrow n} \propto \exp(\psi_\alpha(m, n)/M), \quad [18.44]$$

9808 where M is the length of the input. This encourages the attention to be more evenly
 9809 dispersed across the input. Self-attention is applied across multiple “heads”, each using
 9810 different projections of $\mathbf{h}^{(i-1)}$ to form the keys, values, and queries.

9811 The output of the self-attentional layer is the representation $\mathbf{z}_m^{(i)}$, which is then passed
 9812 through a two-layer feed-forward network, yielding the input to the next layer, $\mathbf{h}^{(i)}$. To
 9813 ensure that information about word order in the source is integrated into the model, the
 9814 encoder includes positional encodings of the index of each word in the source. These
 9815 encodings are vectors for each position $m \in \{1, 2, \dots, M\}$. The positional encodings are
 9816 concatenated with the word embeddings \mathbf{x}_m at the base layer of the model.⁶

9817 Convolutional neural networks (see § 3.4) have also been applied as encoders in neural
 9818 machine translation. For each word $w_m^{(s)}$, a convolutional network computes a representation
 9819 $\mathbf{h}_m^{(s)}$ from the embeddings of the word and its neighbors. This procedure is applied several
 9820 times, creating a deep convolutional network. The recurrent decoder then computes a set
 9821 of attention weights over these convolutional representations, using the decoder’s hidden
 9822 state $\mathbf{h}^{(t)}$ as the queries. This attention vector is used to compute a weighted average over
 9823 the outputs of another convolutional neural network of the source, yielding an averaged
 9824 representation \mathbf{c}_m , which is then fed into the decoder. As with the transformer, speed is
 9825 the main advantage over recurrent encoding models; another similarity is that word order
 9826 information is approximated through the use of positional encodings. It seems likely that
 9827 there are limitations to how well positional encodings can account for word order and
 9828 deeper linguistic structure. But for the moment, the computational advantages of such
 9829 approaches have put them on par with the best recurrent translation models.⁷

9830 18.3.3 Out-of-vocabulary words

9831 Thus far, we have treated translation as a problem at the level of words or phrases. For
 9832 words that do not appear in the training data, all such models will struggle. There are two
 9833 main reasons for the presence of out-of-vocabulary (OOV) words:

- 9834 • New proper nouns, such as family names or organizations, are constantly arising —
 9835 particularly in the news domain. The same is true, to a lesser extent, for technical
 9836 terminology. This issue is shown in Figure 18.7.
- 9837 • In many languages, words have complex internal structure, known as morphology. An
 9838 example is German, which uses compounding to form nouns like Abwasserbehandlungsanlage
 9839 (sewage water treatment plant; example from Sennrich et al. (2016)). While compounds

⁶The transformer architecture relies on several additional tricks, including layer normalization (see § 3.3.4) and residual connections around the nonlinear activations (see § 3.2.2).

⁷A recent evaluation found that best performance was obtained by using a recurrent network for the decoder, and a transformer for the encoder (Chen et al., 2018). The transformer was also found to significantly outperform a convolutional neural network.

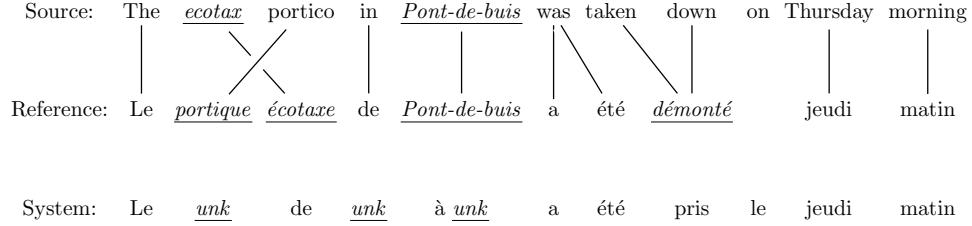


Figure 18.7: Translation with unknown words. The system outputs unk to indicate words that are outside its vocabulary. Figure adapted from Luong et al. (2015).

9840 could in principle be addressed by better tokenization (see § 8.4), other morphological
9841 processes involve more complex transformations of subword units.

9842 Names and technical terms can be handled in a postprocessing step: after first identifying
9843 alignments between unknown words in the source and target, we can look up each aligned
9844 source word in a dictionary, and choose a replacement (Luong et al., 2015). If the word
9845 does not appear in the dictionary, it is likely to be a proper noun, and can be copied
9846 directly from the source to the target. This approach can also be integrated directly into
9847 the translation model, rather than applying it as a postprocessing step (Jean et al., 2015).

9848 Words with complex internal structure can be handled by translating subword units
9849 rather than entire words. A popular technique for identifying subword units is byte-pair
9850 encoding (BPE; Gage, 1994; Sennrich et al., 2016). The initial vocabulary is defined as the
9851 set of characters used in the text. The most common character bigram is then merged into
9852 a new symbol, and the vocabulary is updated. The merging operation is applied repeatedly,
9853 until the vocabulary reaches some maximum size. For example, given the dictionary
9854 {fish, fished, want, wanted, bike, biked}, we would first merge e+d into the subword unit
9855 ed, since this bigram appears in three words of the six words. Next, there are several
9856 bigrams that each appear in a pair of words: f+i, i+s, s+h, w+a, a+n, etc. These can be
9857 merged in any order, resulting in the segmentation, {fish, fish+ed, want, want+ed, bik+e, bik+ed}.
9858 At this point, there are no subword bigrams that appear more than once. In real data,
9859 merging is performed until the number of subword units reaches some predefined threshold,
9860 such as 10^4 .

9861 Each subword unit is treated as a token for translation, in both the encoder (source
9862 side) and decoder (target side). BPE can be applied jointly to the union of the source
9863 and target vocabularies, identifying subword units that appear in both languages. For
9864 languages that have different scripts, such as English and Russian, transliteration between
9865 the scripts should be applied first.⁸

⁸Transliteration is crucial for converting names and other foreign words between languages that do not

9866 18.4 Decoding

Given a trained translation model, the decoding task is:

$$\hat{\mathbf{w}}^{(t)} = \underset{\mathbf{w} \in \mathcal{V}^*}{\operatorname{argmax}} \Psi(\mathbf{w}, \mathbf{w}^{(s)}), \quad [18.45]$$

9867 where $\mathbf{w}^{(t)}$ is a sequence of tokens from the target vocabulary \mathcal{V} . It is not possible to
 9868 efficiently obtain exact solutions to the decoding problem, for even minimally effective
 9869 models in either statistical or neural machine translation. Today's state-of-the-art translation
 9870 systems use beam search (see § 11.3.1.4), which is an incremental decoding algorithm that
 9871 maintains a small constant number of competitive hypotheses. Such greedy approximations
 9872 are reasonably effective in practice, and this may be in part because the decoding objective
 9873 is only loosely correlated with measures of translation quality, so that exact optimization
 9874 of [18.45] may not greatly improve the resulting translations.

Decoding in neural machine translation is somewhat simpler than in phrase-based
 statistical machine translation.⁹ The scoring function Ψ is defined,

$$\Psi(\mathbf{w}^{(t)}, \mathbf{w}^{(s)}) = \sum_{m=1}^{M^{(t)}} \psi(w_m^{(t)}; \mathbf{w}_{1:m-1}^{(t)}, \mathbf{z}) \quad [18.46]$$

$$\psi(w^{(t)}; \mathbf{w}_{1:m-1}^{(t)}, \mathbf{z}) = \beta_{w_m^{(t)}} \cdot \mathbf{h}_m^{(t)} - \log \sum_{w \in \mathcal{V}} \exp(\beta_w \cdot \mathbf{h}_m^{(t)}), \quad [18.47]$$

9875 where \mathbf{z} is the encoding of the source sentence $\mathbf{w}^{(s)}$, and $\mathbf{h}_m^{(t)}$ is a function of the encoding
 9876 \mathbf{z} and the decoding history $\mathbf{w}_{1:m-1}^{(t)}$. This formulation subsumes the attentional translation
 9877 model, where \mathbf{z} is a matrix encoding of the source.

Now consider the incremental decoding algorithm,

$$\hat{w}_m^{(t)} = \underset{w \in \mathcal{V}}{\operatorname{argmax}} \psi(w; \hat{\mathbf{w}}_{1:m-1}^{(t)}, \mathbf{z}), \quad m = 1, 2, \dots \quad [18.48]$$

9878 This algorithm selects the best target language word at position m , assuming that it has
 9879 already generated the sequence $\hat{\mathbf{w}}_{1:m-1}^{(t)}$. (Termination can be handled by augmenting the
 9880 vocabulary \mathcal{V} with a special end-of-sequence token, ■.) The incremental algorithm is likely
 9881 to produce a suboptimal solution to the optimization problem defined in Equation 18.45,
 9882 because selecting the highest-scoring word at position m can set the decoder on a “garden
 9883 path,” in which there are no good choices at some later position $n > m$. We might hope
 9884 for some dynamic programming solution, as in sequence labeling (§ 7.3). But the Viterbi

share a single script, such as English and Japanese. It is typically approached using the finite-state methods discussed in chapter 9 (Knight and Graehl, 1998).

⁹For more on decoding in phrase-based statistical models, see Koehn (2009).

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 $\mathbf{y}_{1:m}$. 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.¹⁰ 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 18.48, but at each step m , a set of K different hypotheses are kept on the beam. For each hypothesis $k \in \{1, 2, \dots, K\}$, 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 the current hidden state $\mathbf{h}_k^{(t)}$. At each step in the beam search, the K top-scoring children of each hypothesis currently on the beam are “expanded”, and the beam is updated. For a detailed description of beam search for RNN decoding, see Graves (2012).

Learning and search Conventionally, the learning algorithm is trained to predict the right token in the translation, conditioned on the translation history being correct. But if decoding must be approximate, then we might do better by modifying the learning algorithm to be robust to errors in the translation history. Scheduled sampling does this by training on histories that sometimes come from the ground truth, and sometimes come from the model’s own output (Bengio et al., 2015).¹¹ As training proceeds, the training wheels come off: we increase the fraction of tokens that come from the model rather than the ground truth. Another approach is to train on an objective that relates directly to beam search performance (Wiseman et al., 2016). Reinforcement learning has also been applied to decoding of RNN-based translation models, making it possible to directly optimize translation metrics such as Bleu (Ranzato et al., 2016).

18.5 Training towards the evaluation metric

In likelihood-based training, the objective is to maximize the probability of a parallel corpus. However, translations are not evaluated in terms of likelihood: metrics like bleu consider only the correctness of a single output translation, and not the range of probabilities that the model assigns. It might therefore be better to train translation models to achieve the highest bleu score possible — to the extent that we believe bleu

¹⁰Note that this problem does not impact RNN-based sequence labeling models (see § 7.6). This is because the tags produced by these models do not affect the recurrent state.

¹¹Scheduled sampling builds on earlier work on learning to search (Daumé III et al., 2009; Ross et al., 2011), which are also described in § 15.2.4.

measures translation quality. Unfortunately, bleu and related metrics are not friendly for optimization: they are discontinuous, non-differentiable functions of the parameters of the translation model.

Consider an error function $\Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)})$, which measures the discrepancy between the system translation $\hat{\mathbf{w}}^{(t)}$ and the reference translation $\mathbf{w}^{(t)}$; this function could be based on bleu or any other metric on translation quality. One possible criterion would be to select the parameters θ that minimize the error of the system's preferred translation,

$$\hat{\mathbf{w}}^{(t)} = \operatorname{argmax}_{\mathbf{w}^{(t)}} \Psi(\mathbf{w}^{(t)}, \mathbf{w}^{(s)}; \theta) \quad [18.49]$$

$$\hat{\theta} = \operatorname{argmin}_{\theta} \Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(s)}) \quad [18.50]$$

However, identifying the top-scoring translation $\hat{\mathbf{w}}^{(t)}$ is usually intractable, as described in the previous section. In minimum error-rate training (MERT), $\hat{\mathbf{w}}^{(t)}$ is selected from a set of candidate translations $\mathcal{Y}(\mathbf{w}^{(s)})$; this is typically a strict subset of all possible translations, so that it is only possible to optimize an approximation to the true error rate (Och and Ney, 2003).

A further issue is that the objective function in Equation 18.50 is discontinuous and non-differentiable, due to the argmax over translations: an infinitesimal change in the parameters θ could cause another translation to be selected, with a completely different error. To address this issue, we can instead minimize the risk, which is defined as the expected error rate,

$$R(\theta) = E_{\hat{\mathbf{w}}^{(t)} | \mathbf{w}^{(s)}; \theta} [\Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)})] \quad [18.51]$$

$$= \sum_{\hat{\mathbf{w}}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})} p(\hat{\mathbf{w}}^{(t)} | \mathbf{w}^{(s)}) \times \Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)}). \quad [18.52]$$

Minimum risk training minimizes the sum of $R(\theta)$ across all instances in the training set.

The risk can be generalized by exponentiating the translation probabilities,

$$\tilde{p}(\mathbf{w}^{(t)}; \theta, \alpha) \propto \left(p(\mathbf{w}^{(t)} | \mathbf{w}^{(s)}; \theta) \right)^\alpha \quad [18.53]$$

$$\tilde{R}(\theta) = \sum_{\hat{\mathbf{w}}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})} \tilde{p}(\hat{\mathbf{w}}^{(t)} | \mathbf{w}^{(s)}; \alpha, \theta) \times \Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)}) \quad [18.54]$$

where $\mathcal{Y}(\mathbf{w}^{(s)})$ is now the set of all possible translations for $\mathbf{w}^{(s)}$. Exponentiating the probabilities in this way is known as annealing (Smith and Eisner, 2006). When $\alpha = 1$, then $\tilde{R}(\theta) = R(\theta)$; when $\alpha = \infty$, then $\tilde{R}(\theta)$ is equivalent to the sum of the errors of the maximum probability translations for each sentence in the dataset.

Clearly the set of candidate translations $\mathcal{Y}(\mathbf{w}^{(s)})$ is too large to explicitly sum over. Because the error function Δ generally does not decompose into smaller parts, there is no

efficient dynamic programming solution to sum over this set. We can approximate the sum $\sum_{\tilde{\mathbf{w}}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})}$ with a sum over a finite number of samples, $\{\mathbf{w}_1^{(t)}, \mathbf{w}_2^{(t)}, \dots, \mathbf{w}_K^{(t)}\}$. If these samples were drawn uniformly at random, then the (annealed) risk would be approximated as (Shen et al., 2016),

$$\tilde{R}(\boldsymbol{\theta}) \approx \frac{1}{Z} \sum_{k=1}^K \tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)}; \boldsymbol{\theta}, \alpha) \times \Delta(\mathbf{w}_k^{(t)}, \mathbf{w}^{(t)}) \quad [18.55]$$

$$Z = \sum_{k=1}^K \tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)}; \boldsymbol{\theta}, \alpha). \quad [18.56]$$

9930 Shen et al. (2016) report that performance plateaus at $K = 100$ for minimum risk training
9931 of neural machine translation.

Uniform sampling over the set of all possible translations is undesirable, because most translations have very low probability. A solution from Monte Carlo estimation is importance sampling, in which we draw samples from a proposal distribution $q(\mathbf{w}^{(t)})$. This distribution can be set equal to the current translation model $p(\mathbf{w}^{(t)} | \mathbf{w}^{(s)}; \boldsymbol{\theta})$. Each sample is then weighted by an importance score, $\omega_k = \frac{\tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)})}{q(\mathbf{w}_k^{(t)})}$. The effect of this weighting is to correct for any mismatch between the proposal distribution q and the true distribution \tilde{p} . The risk can then be approximated as,

$$\mathbf{w}_k^{(t)} \sim q(\mathbf{w}^{(t)}) \quad [18.57]$$

$$\omega_k = \frac{\tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)})}{q(\mathbf{w}_k^{(t)})} \quad [18.58]$$

$$\tilde{R}(\boldsymbol{\theta}) \approx \frac{1}{\sum_{k=1}^K \omega_k} \sum_{k=1}^K \omega_k \times \Delta(\mathbf{w}_k^{(t)}, \mathbf{w}^{(t)}). \quad [18.59]$$

9932 Importance sampling will generally give a more accurate approximation with a given
9933 number of samples. The only formal requirement is that the proposal assigns non-zero
9934 probability to every $\mathbf{w}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})$. For more on importance sampling and related
9935 methods, see Robert and Casella (2013).

9936 Additional resources

9937 A complete textbook on machine translation is available from Koehn (2009). While this
9938 book precedes recent work on neural translation, a more recent draft chapter on neural
9939 translation models is also available (Koehn, 2017). Neubig (2017) provides a comprehensive
9940 tutorial on neural machine translation, starting from first principles. The course notes from
9941 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 primarily in Torch; 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. Messud (2014) describes some of these issues in her review of an English translation of *L'étranger*, the 1942 French novel by Albert Camus.¹² 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:

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.

¹²The book review is currently available online at <http://www.nybooks.com/articles/2014/06/05/camus-new-letranger/>.

9974 Exercises

9975 1. Give a synchronized derivation (§ 18.2.4) for the Spanish-English translation,

9976 (18.4) El pez enojado atacado.
The fish angry attacked.
9977 The angry fish attacked.

9978 As above, the second line shows a word-for-word gloss, and the third line shows
9979 the desired translation. Use the synchronized production rule in [18.22], and design
9980 the other production rules necessary to derive this sentence pair. You may derive
9981 (atacado, attacked) directly from VP.

9982 Chapter 19

9983 Text generation

9984 In many of the most interesting problems in natural language processing, language is
9985 the output. The previous chapter described the specific case of machine translation, but
9986 there are many other applications, from summarization of research articles, to automated
9987 journalism, to dialogue systems. This chapter emphasizes three main scenarios: data-to-text,
9988 in which text is generated to explain or describe a structured record or unstructured
9989 perceptual input; text-to-text, which typically involves fusing information from multiple
9990 linguistic sources into a single coherent summary; and dialogue, in which text is generated
9991 as part of an interactive conversation with one or more human participants.

9992 19.1 Data-to-text generation

9993 In data-to-text generation, the input ranges from structured records, such as the description
9994 of a weather forecast (as shown in Figure 19.1), to unstructured perceptual data, such as
9995 a raw image or video; the output may be a single sentence, such as an image caption, or
9996 a multi-paragraph argument. Despite this diversity of conditions, all data-to-text systems
9997 share some of the same challenges (Reiter and Dale, 2000):

- 9998 • determining what parts of the data to describe;
9999 • planning a presentation of this information;
10000 • lexicalizing the data into words and phrases;
10001 • organizing words and phrases into well-formed sentences and paragraphs.

10002 The earlier stages of this process are sometimes called content selection and text planning;
10003 the later stages are often called surface realization.

10004 Early systems for data-to-text generation were modular, with separate software components
10005 for each task. Artificial intelligence planning algorithms can be applied to both the

Database:	Temperature			Cloud Sky Cover		
	time	min	mean	max	time	percent (%)
	06:00-21:00	9	15	21	06:00-09:00	25-50
					09:00-12:00	50-75
Wind Speed			Wind Direction			
time			time			
	06:00-21:00	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]

10006 high-level information structure and the organization of individual sentences, ensuring that
 10007 communicative goals are met (McKeown, 1992; Moore and Paris, 1993). Surface realization
 10008 can be performed by grammars or templates, which link specific types of data to candidate
 10009 words and phrases. A simple example template is offered by Wiseman et al. (2017), for
 10010 generating descriptions of basketball games:

10011 (19.1) The <team1> (<wins1>-losses1) defeated the <team2> (<wins2>-<losses2>),
 10012 <pts1>-<pts2>.
 10013 The New York Knicks (45-5) defeated the Boston Celtics (11-38), 115-79.

10014 For more complex cases, it may be necessary to apply morphological inflections such as
 10015 pluralization and tense marking — even in the simple example above, languages such as
 10016 Russian would require case marking suffixes for the team names. Such inflections can be
 10017 applied as a postprocessing step. Another difficult challenge for surface realization is the
 10018 generation of varied referring expressions (e.g., The Knicks, New York, they), which is
 10019 critical to avoid repetition. As discussed in § 16.2.1, the form of referring expressions is
 10020 constrained by the discourse and information structure.

10021 An example at the intersection of rule-based and statistical techniques is the Nitrogen
 10022 system (Langkilde and Knight, 1998). The input to Nitrogen is an abstract meaning
 10023 representation (AMR; see § 13.3) of semantic content to be expressed in a single sentence.
 10024 In data-to-text scenarios, the abstract meaning representation is the output of a higher-level
 10025 text planning stage. A set of rules then converts the abstract meaning representation into
 10026 various sentence plans, which may differ in both the high-level structure (e.g., active versus
 10027 passive voice) as well as the low-level details (e.g., word and phrase choice). Some examples
 10028 are shown in Figure 19.2. To control the combinatorial explosion in the number of possible
 10029 realizations for any given meaning, the sentence plans are unified into a single finite-state
 10030 acceptor, in which word tokens are represented by arcs (see § 9.1.1). A bigram language

```
(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).

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

$$\hat{\mathbf{w}} = \underset{\mathbf{w} \in \mathcal{V}^*}{\operatorname{argmax}} \Psi(\mathbf{w}, \mathbf{y}; \boldsymbol{\theta}), \quad [19.1]$$

where \mathcal{V}^* is the set of strings over a discrete vocabulary, and $\boldsymbol{\theta}$ is a vector of parameters. The relationship between \mathbf{w} and \mathbf{y} is complex: the data \mathbf{y} may contain dozens of records, and \mathbf{w} may extend to several sentences. To facilitate learning and inference, it would be helpful to decompose the scoring function Ψ into subcomponents. This would be possible if given an alignment, specifying which element of \mathbf{y} is expressed in each part of \mathbf{w} (Angeli et al., 2010):

$$\Psi(\mathbf{w}, \mathbf{y}; \boldsymbol{\theta}) = \sum_{m=1}^M \psi_{w,y}(\mathbf{w}_m, \mathbf{y}_{z_m}) + \psi_z(z_m, z_{m-1}), \quad [19.2]$$

where z_m indicates the record aligned to word m . For example, in Figure 19.1, z_1 might specify that the word cloudy is aligned to the record cloud-sky-cover:percent. The score for this alignment would then be given by the weight on features such as

$$(\text{cloudy}, \text{cloud-sky-cover:percent}), \quad [19.3]$$

10053 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
 10054 to assign higher scores to alignments that are coherent, referring to the same records in
 10055 adjacent parts of the text.¹

10056 Several datasets include structured records and natural language text (Barzilay and
 10057 McKeown, 2005; Chen and Mooney, 2008; Liang and Klein, 2009), but the alignments
 10058 between text and records are usually not available.² One solution is to model the problem
 10059 probabilistically, treating the alignment as a latent variable (Liang et al., 2009; Konstas
 10060 and Lapata, 2013). The model can then be estimated using expectation maximization or
 10061 sampling (see chapter 5).

10062 19.1.2 Neural data-to-text generation

10063 The encoder-decoder model and neural attention were introduced in § 18.3 as methods
 10064 for neural machine translation. They can also be applied to data-to-text generation, with
 10065 the data acting as the source language (Mei et al., 2016). In neural machine translation,
 10066 the attention mechanism linked words in the source to words in the target; in data-to-text
 10067 generation, the attention mechanism can link each part of the generated text back to a
 10068 record in the data. The biggest departure from translation is in the encoder, which depends
 10069 on the form of the data.

10070 19.1.2.1 Data encoders

10071 In some types of structured records, all values are drawn from discrete sets. For example,
 10072 the birthplace of an individual is drawn from a discrete set of possible locations; the
 10073 diagnosis and treatment of a patient are drawn from an exhaustive list of clinical codes (Johnson
 10074 et al., 2016). In such cases, vector embeddings can be estimated for each field and possible
 10075 value: for example, a vector embedding for the field birthplace, and another embedding for
 10076 the value Berkeley_California (Bordes et al., 2011). The table of such embeddings serves
 10077 as the encoding of a structured record (He et al., 2017). It is also possible to compress the
 10078 entire table into a single vector representation, by pooling across the embeddings of each
 10079 field and value (Lebret et al., 2016).

Sequences Some types of structured records have a natural ordering, such as events in a game (Chen and Mooney, 2008) and steps in a recipe (Tutin and Kittredge, 1992). For example, the following records describe a sequence of events in a robot soccer match (Mei

¹More expressive decompositions of Ψ are possible. For example, Wong and Mooney (2007) use a synchronous context-free grammar (see § 18.2.4) to “translate” between a meaning representation and natural language text.

²An exception is a dataset of records and summaries from American football games, containing annotations of alignments between sentences and records (Snyder and Barzilay, 2007).



Figure 19.3: Examples of the image captioning task, with attention masks shown for each of the underlined words. From Xu et al. (2015). [todo: permission]

et al., 2016):

```
pass(arg1 = purple6, arg2 = purple3)
kick(arg1 = purple3)
badpass(arg1 = purple3, arg2 = pink9).
```

10080 Each event is a single record, and can be encoded by a concatenation of vector representations
 10081 for the event type (e.g., pass), the field (e.g., arg1), and the values (e.g., purple3), e.g.,

$$\mathbf{X} = [\mathbf{u}_{\text{pass}}, \mathbf{u}_{\text{arg1}}, \mathbf{u}_{\text{purple6}}, \mathbf{u}_{\text{arg2}}, \mathbf{u}_{\text{purple3}}]. \quad [19.4]$$

10082 This encoding can then act as the input layer for a recurrent neural network, yielding a
 10083 sequence of vector representations $\{\mathbf{z}_r\}_{r=1}^R$, where r indexes over records. Interestingly,
 10084 this sequence-based approach is effective even in cases where there is no natural ordering
 10085 over the records, such as the weather data in Figure 19.1 (Mei et al., 2016).

10086 **Images** Another flavor of data-to-text generation is the generation of text captions for
 10087 images. Examples from this task are shown in Figure 19.3. Images are naturally represented
 10088 as tensors: a color image of 320×240 pixels would be stored as a tensor with $320 \times$
 10089 240×3 intensity values. The dominant approach to image classification is to encode
 10090 images as vectors using a combination of convolution and pooling (Krizhevsky et al., 2012).
 10091 Chapter 3 explains how to use convolutional networks for text; for images, convolution is
 10092 applied across the vertical, horizontal, and color dimensions. By pooling the results of
 10093 successive convolutions, the image is converted to a vector representation, which can then

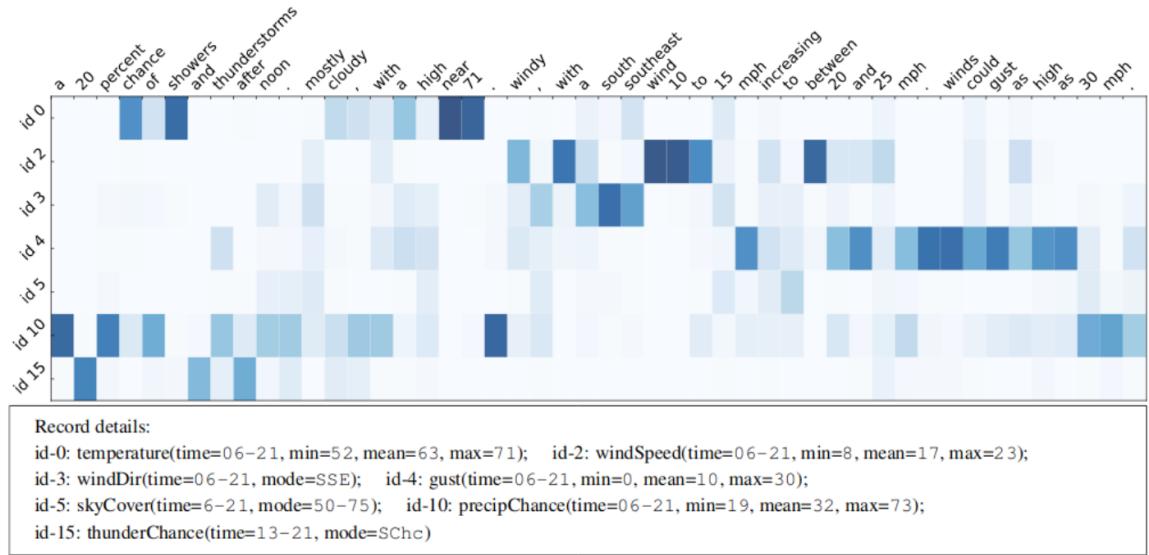


Figure 19.4: Neural attention in text generation. Figure from Mei et al. (2016).[todo: permission]

10094 be fed directly into the decoder as the initial state (Vinyals et al., 2015), just as in the
 10095 sequence-to-sequence translation model (see § 18.3). Alternatively, one can apply a set
 10096 of convolutional networks, yielding vector representations for different parts of the image,
 10097 which can then be combined using neural attention (Xu et al., 2015).

10098 19.1.2.2 Attention

Given a set of embeddings of the data $\{\mathbf{z}_r\}_{r=1}^R$ and a decoder state \mathbf{h}_m , the attention vector over the data can be computed using the same technique described in § 18.3.1. When generating word m of the output, a softmax attention mechanism computes the weighted average \mathbf{c}_m ,

$$\psi_\alpha(m, r) = \beta_\alpha \cdot f(\Theta_\alpha[\mathbf{h}_m; \mathbf{z}_r]) \quad [19.5]$$

$$\boldsymbol{\alpha}_m = \text{SoftMax}([\psi_\alpha(m, 1), \psi_\alpha(m, 2), \dots, \psi_\alpha(m, R)]) \quad [19.6]$$

$$\mathbf{c}_m = \sum_{r=1}^R \alpha_{m \rightarrow r} \mathbf{z}_r, \quad [19.7]$$

10099 where f is an elementwise nonlinearity such as tanh or ReLU (see § 3.2.1). The weighted
 10100 average \mathbf{c}_m can then be included in the recurrent update to the decoder state, or in
 10101 the emission probabilities, as described in § 18.3.1. Figure 19.4 shows the attention to
 10102 components of a weather record, while generating the text shown on the x -axis.

10103 Adapting this architecture to image captioning is straightforward. A convolutional
 10104 neural networks is applied to a set of image locations, and the output at each location ℓ is
 10105 represented with a vector \mathbf{z}_ℓ . Attention can then be computed over the image locations,
 10106 as shown in the right panels of each pair of images in Figure 19.3.

10107 Various modifications to this basic mechanism have been proposed. In coarse-to-fine
 10108 attention (Mei et al., 2016), each record receives a global attention $a_r \in [0, 1]$, which
 10109 is independent of the decoder state. This global attention, which represents the overall
 10110 importance of the record, is multiplied with the decoder-based attention scores, before
 10111 computing the final normalized attentions. In structured attention, the attention vector
 10112 $\boldsymbol{\alpha}_{m \rightarrow \cdot}$ can include structural biases, which can favor assigning higher attention values to
 10113 contiguous segments or to dependency subtrees (Kim et al., 2017). Structured attention
 10114 vectors can be computed by running the forward-backward algorithm to obtain marginal
 10115 attention probabilities (see § 7.5.3.3). Because each step in the forward-backward algorithm
 10116 is differentiable, it can be encoded in a computation graph, and end-to-end learning can
 10117 be performed by backpropagation.

10118 19.1.2.3 Decoder

10119 Given the encoding, the decoder can function just as in neural machine translation (see
 10120 § 18.3.1), using the attention-weighted encoder representation in the decoder recurrence
 10121 and/or output computation. As in machine translation, beam search can help to avoid
 10122 search errors (Lebret et al., 2016).

Many applications require generating words that do not appear in the training vocabulary. For example, a weather record may contain a previously unseen city name; a sports record may contain a previously unseen player name. Such tokens can be generated in the text by copying them over from the input (e.g., Gulcehre et al., 2016).³ First introduce an additional variable $s_m \in \{\text{gen}, \text{copy}\}$, indicating whether token $w_m^{(t)}$ should be generated or copied. The decoder probability is then,

$$p(w^{(t)} | \mathbf{w}_{1:m-1}^{(t)}, \mathbf{Z}, s_m) = \begin{cases} \text{SoftMax}(\boldsymbol{\beta}_{w^{(t)}} \cdot \mathbf{h}_{m-1}^{(t)}), & s_m = \text{gen} \\ \sum_{r=1}^R \delta(w_r^{(s)} = w^{(t)}) \times \alpha_{m \rightarrow r}, & s_m = \text{copy}, \end{cases} \quad [19.8]$$

10123 where $\delta(w_r^{(s)} = w^{(t)})$ is an indicator function, taking the value 1 iff the text of the record
 10124 $w_r^{(s)}$ is identical to the target word $w^{(t)}$. The probability of copying record r from the source
 10125 is $\delta(s_m = \text{copy}) \times \alpha_{m \rightarrow r}$, the product of the copy probability by the local attention. Note
 10126 that in this model, the attention weights $\boldsymbol{\alpha}_m$ are computed from the previous decoder state
 10127 \mathbf{h}_{m-1} . The computation graph therefore remains a feedforward network, with recurrent
 10128 paths such as $\mathbf{h}_{m-1}^{(t)} \rightarrow \boldsymbol{\alpha}_m \rightarrow w_m^{(t)} \rightarrow \mathbf{h}_m^{(t)}$.

³A number of variants of this strategy have been proposed (e.g., Gu et al., 2016; Merity et al., 2017). See Wiseman et al. (2017) for an overview.

10129 To facilitate end-to-end training, the switching variable s_m can be represented by a
 10130 gate π_m , which is computed from a two-layer feedforward network, whose input consists of
 10131 the concatenation of the decoder state $\mathbf{h}_{m-1}^{(t)}$ and the attention-weighted representation of
 10132 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,

$$p(w^{(t)} | \mathbf{w}_{1:m}^{(t)}, \mathbf{Z}) = \pi_m \times \underbrace{\frac{\exp \beta_{w^{(t)}} \cdot \mathbf{h}_{m-1}^{(t)}}{\sum_{j=1}^V \exp \beta_j \cdot \mathbf{h}_{m-1}^{(t)}}}_{\text{generate}} + (1 - \pi_m) \times \underbrace{\sum_{r=1}^R \delta(w_r^{(s)} = w^{(t)}) \times \alpha_{m \rightarrow r}}_{\text{copy}}. \quad [19.10]$$

10133 19.2 Text-to-text generation

10134 Text-to-text generation includes problems of summarization and simplification:

- 10135 • reading a novel and outputting a paragraph-long summary of the plot;⁴
- 10136 • reading a set of blog posts about politics, and outputting a bullet list of the various
 10137 issues and perspectives;
- 10138 • reading a technical research article about the long-term health consequences of drinking
 10139 kombucha, and outputting a summary of the article in language that non-experts can
 10140 understand.

10141 These problems can be approached in two ways: through the encoder-decoder architecture
 10142 discussed in the previous section, or by operating directly on the input text.

10143 19.2.1 Neural abstractive summarization

10144 Sentence summarization is the task of shortening a sentence while preserving its meaning,
 10145 as in the following examples (Knight and Marcu, 2000; Rush et al., 2015):

- 10146 (19.2) The documentation is typical of Epson quality: excellent.
 10147 Documentation is excellent.

⁴In § 16.3.4.1, we encountered a special case of single-document summarization, which involved extracting the most important sentences or discourse units. We now consider the more challenging problem of abstractive summarization, in which the summary can include words that do not appear in the original text.

- 10149 (19.3) Russian defense minister Ivanov called sunday for the creation of a joint front for
 10150 combating global terrorism.
 10151 Russia calls for joint front against terrorism.
 10152

10153 Sentence summarization is closely related to sentence compression, in which the summary
 10154 is produced by deleting words or phrases from the original (Clarke and Lapata, 2008). But
 10155 as shown in (19.3), a sentence summary can also introduce new words, such as against,
 10156 which replaces the phrase for combatting.

10157 Sentence summarization can be treated as a machine translation problem, using the
 10158 attentional encoder-decoder translation model discussed in § 18.3.1 (Rush et al., 2015).
 10159 The longer sentence is encoded into a sequence of vectors, one for each token. The decoder
 10160 then computes attention over these vectors when updating its own recurrent state. As
 10161 with data-to-text generation, it can be useful to augment the encoder-decoder model with
 10162 the ability to copy words directly from the source. Rush et al. (2015) train this model by
 10163 building four million sentence pairs from news articles. In each pair, the longer sentence
 10164 is the first sentence of the article, and the summary is the article headline. Sentence
 10165 summarization can also be trained in a semi-supervised fashion, using a probabilistic
 10166 formulation of the encoder-decoder model called a variational autoencoder (Miao and
 10167 Blunsom, 2016, 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, $\mathbf{t}_m = \sum_{n=1}^m \boldsymbol{\alpha}_n$. 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,

$$\ell_m = \sum_{n=1}^{M^{(s)}} \min(\alpha_{m \rightarrow n}, t_{m \rightarrow n}). \quad [19.12]$$

10168 This loss will be low if $\boldsymbol{\alpha}_{m \rightarrow ..}$ assigns little attention to words that already have large
 10169 values in $\mathbf{t}_{m \rightarrow ..}$. Coverage loss is similar to the concept of marginal relevance, in which
 10170 the reward for adding new content is proportional to the extent to which it increases the
 10171 overall amount of information conveyed by the summary (Carbonell and Goldstein, 1998).

10172 19.2.2 Sentence fusion for multi-document summarization

10173 In multi-document summarization, the goal is to produce a summary that covers the
 10174 content of several documents (McKeown et al., 2002). One approach to this challenging
 10175 problem is to identify sentences across multiple documents that relate to a single theme,
 10176 and then to fuse them into a single sentence (Barzilay and McKeown, 2005). As an example,
 10177 consider the following two sentences (McKeown et al., 2010):

- 10178 (19.4) Palin actually turned against the bridge project only after it became a national
 10179 symbol of wasteful spending.
- 10180 (19.5) Ms. Palin supported the bridge project while running for governor, and abandoned
 10181 it after it became a national scandal.

10182 An intersection preserves only the content that is present in both sentences:

- 10183 (19.6) Palin turned against the bridge project after it became a national scandal.

10184 A union includes information from both sentences:

- 10185 (19.7) Ms. Palin supported the bridge project while running for governor, but turned
 10186 against it when it became a national scandal and a symbol of wasteful spending.

Dependency parsing is often used as a technique for sentence fusion. After parsing each sentence, the resulting dependency trees can be aggregated into a lattice (Barzilay and McKeown, 2005) or a graph structure (Filippova and Strube, 2008), in which identical or closely related words (e.g., Palin, bridge, national) are fused into a single node. The resulting graph can then be pruned back to a tree by solving an integer linear program (see § 13.2.2),

$$\max_{\mathbf{y}} \sum_{i,j,r} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) \times y_{i,j,r} \quad [19.13]$$

$$\text{s.t. } \mathbf{y} \in \mathcal{C}, \quad [19.14]$$

10187 where the variable $y_{i,j,r} \in \{0, 1\}$ indicates whether there is an edge from i to j of type r , the
 10188 score of this edge is $\psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})$, and \mathcal{C} is a set of constraints, described below. As usual,
 10189 \mathbf{w} is the list of words in the graph, and $\boldsymbol{\theta}$ is a vector of parameters. The score $\psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})$
 10190 reflects the “importance” of the modifier j to the overall meaning: in intersective fusion,
 10191 this score indicates the extent to which the content in this edge is expressed in all sentences;
 10192 in union fusion, the score indicates whether the content in the edge is expressed in any
 10193 sentence.

10194 The constraint set \mathcal{C} ensures that \mathbf{y} forms a valid dependency graph. It can also
 10195 impose additional linguistic constraints: for example, ensuring that coordinated nouns are

10196 sufficiently similar. The resulting tree must then be linearized into a sentence. This is
10197 typically done by generating a set of candidate linearizations, and choosing the one with
10198 the highest score under a language model (Langkilde and Knight, 1998; Song et al., 2016).

10199

19.3 Dialogue

10200 Dialogue systems are capable of conversing with a human interlocutor, often to perform
10201 some task (Grosz, 1979), but sometimes just to chat (Weizenbaum, 1966). While research
10202 on dialogue systems goes back several decades (Carbonell, 1970; Winograd, 1972), commercial
10203 systems such as Alexa and Siri have recently brought this technology into widespread
10204 use. Nonetheless, there is a significant gap between research and practice: many practical
10205 dialogue systems remain scripted and inflexible, while research systems emphasize abstractive
10206 text generation, “on-the-fly” decision making, and probabilistic reasoning about the user’s
10207 intentions.

10208

19.3.1 Finite-state and agenda-based dialogue systems

10209 Finite-state automata were introduced in chapter 9 as a formal model of computation, in
10210 which string inputs and outputs are linked to transitions between a finite number of discrete
10211 states. This model naturally fits simple task-oriented dialogues, such as the one shown in
10212 the left panel of Figure 19.5. This (somewhat frustrating) dialogue can be represented with
10213 a finite-state transducer, as shown in the right panel of the figure. The accepting state is
10214 reached only when the two needed pieces of information are provided, and the human user
10215 confirms that the order is correct. In this simple scenario, the topping and address are the
10216 two slots associated with the activity of ordering a pizza, which is called a frame. Frame
10217 representations can be hierarchical: for example, an address could have slots of its own,
10218 such as street and city.

10219 In the example dialogue in Figure 19.5, the user provides the precise inputs that are
10220 needed in each turn (e.g., anchovies; the College of Computing building). Some users
10221 may prefer to communicate more naturally, with phrases like I’d, uh, like some anchovies
10222 please. One approach to handling such utterances is to design a custom grammar, with
10223 non-terminals for slots such as topping and location. However, context-free parsing of
10224 unconstrained speech input is challenging. A more lightweight alternative is BIO-style
10225 sequence labeling (see § 8.3), e.g.:

10226 (19.9) I’d like anchovies , and please bring it to the College of
10227 O O B-TOPPING O O O O O O B-ADDR I-ADDR I-ADDR
Computing Building .
I-ADDR I-ADDR O

- (19.8) A: I want to order a pizza.
 B: What toppings?
 A: Anchovies.
 B: Ok, what address?
 A: The College of Computing building.
 B: Please confirm: one pizza with artichokes, to be delivered to the College of Computing building.
 A: No.
 B: What toppings?
 ...

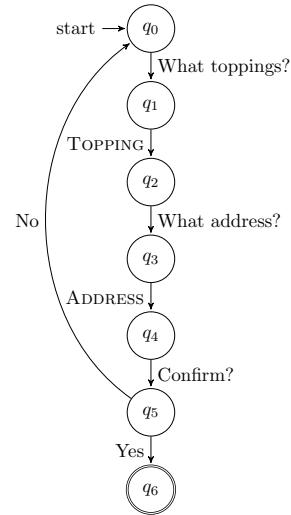


Figure 19.5: An example dialogue and the associated finite-state model. In the finite-state model, small caps indicates that the user must provide information of this type in their answer.

10228 The tagger can be driven by a bi-directional recurrent neural network, similar to recurrent
 10229 approaches to semantic role labeling described in § 13.2.3.

10230 The input in (19.9) could not be handled by the finite-state system from Figure 19.5,
 10231 which forces the user to provide the topping first, and then the location. In this sense,
 10232 the initiative is driven completely by the system. Agenda-based dialogue systems extend
 10233 finite-state architectures by attempting to recognize all slots that are filled by the user’s
 10234 reply, thereby handling these more complex examples. Agenda-based systems dynamically
 10235 pose additional questions until the frame is complete (Bobrow et al., 1977; Allen et al.,
 10236 1995; Rudnicky and Xu, 1999). Such systems are said to be mixed-initiative, because both
 10237 the user and the system can drive the direction of the dialogue.

10238 19.3.2 Markov decision processes

10239 The task of dynamically selecting the next move in a conversation is known as dialogue
 10240 management. This problem can be framed as a Markov decision process, which is a
 10241 theoretical model that includes a discrete set of states, a discrete set of actions, a function
 10242 that computes the probability of transitions between states, and a function that computes
 10243 the cost or reward of action-state pairs. Let’s see how each of these elements pertains to
 10244 the pizza ordering dialogue system.

- 10245 • Each state is a tuple of information about whether the topping and address are

10246 known, and whether the order has been confirmed. For example,

$$(Known\ Topping,\ Unknown\ Address,\ Not\ confirmed) \quad [19.15]$$

10247 is a possible state. Any state in which the pizza order is confirmed is a terminal
 10248 state, and the Markov decision process stops after entering such a state.

- 10249 • The set of actions includes querying for the topping, querying for the address, and
 10250 requesting confirmation. Each action induces a probability distribution over states,
 10251 $p(s_t | a_t, s_{t-1})$. For example, requesting confirmation of the order is not likely to
 10252 result in a transition to the terminal state if the topping is not yet known. This
 10253 probability distribution over state transitions may be learned from data, or it may
 10254 be specified in advance.
- 10255 • Each state-action-state tuple earns a reward, $r_a(s_t, s_{t+1})$. In the context of the pizza
 10256 ordering system, a simple reward function would be,

$$r_a(s_t, s_{t+1}) = \begin{cases} 0, & a = \text{Confirm}, s_{t+1} = (*, *, \text{Confirmed}) \\ -10, & a = \text{Confirm}, s_{t+1} = (*, *, \text{Not Confirmed}) \\ -1, & a \neq \text{Confirm} \end{cases} \quad [19.16]$$

10257 This function assigns zero reward for successful transitions to the terminal state, a
 10258 large negative reward to a rejected request for confirmation, and a small negative
 10259 reward for every other type of action. The system is therefore rewarded for reaching
 10260 the terminal state in few steps, and penalized for prematurely requesting confirmation.

10261 In a Markov decision process, a policy is a function $\pi : \mathcal{S} \mapsto \mathcal{A}$ that maps from states to
 10262 actions (see § 15.2.4.3). The value of a policy is the expected sum of discounted rewards,
 10263 $E_\pi[\sum_{t=1}^T \gamma^t r_{a_t}(s_t, s_{t+1})]$, where γ is the discount factor, $\gamma \in [0, 1)$. Discounting has the
 10264 effect of emphasizing rewards that can be obtained immediately over less certain rewards
 10265 in the distant future.

10266 An optimal policy can be obtained by dynamic programming, by iteratively updating
 10267 the value function $V(s)$, which is the expectation of the cumulative reward from s under
 10268 the optimal action a ,

$$V(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s' | s, a)[r_a(s, s') + \gamma V(s')]. \quad [19.17]$$

10269 The value function $V(s)$ is computed in terms of $V(s')$ for all states $s' \in \mathcal{S}$. A series
 10270 of iterative updates to the value function will eventually converge to a stationary point.
 10271 This algorithm is known as value iteration. Given the converged value function $V(s)$, the
 10272 optimal action at each state is the argmax,

$$\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s' | s, a)[r_a(s, s') + \gamma V(s')]. \quad [19.18]$$

10273 Value iteration and related algorithms are described in detail by Sutton and Barto (1998).
 10274 For applications to dialogue systems, see Levin et al. (1998) and Walker (2000).

10275 The Markov decision process framework assumes that the current state of the dialogue
 10276 is known. In reality, the system may misinterpret the user’s statements — for example,
 10277 believing that a specification of the delivery location (Peachtree) is in fact a specification
 10278 of the topping (peaches). In a partially observable Markov decision process (POMDP),
 10279 the system receives an observation o , which is probabilistically conditioned on the state,
 10280 $p(o | s)$. It must therefore maintain a distribution of beliefs about which state it is in, with
 10281 $q_t(s)$ indicating the degree of belief that the dialogue is in state s at time t . The POMDP
 10282 formulation can help to make dialogue systems more robust to errors, particularly in the
 10283 context of spoken language dialogues, where the speech itself may be misrecognized (Roy
 10284 et al., 2000; Williams and Young, 2007). However, finding the optimal policy in a POMDP
 10285 is computationally intractable, requiring additional approximations.

10286 19.3.3 Neural chatbots

10287 Chatting is a lot easier when you don’t need to get anything done. Chatbots are systems
 10288 that parry the user’s input with a response that keeps the conversation going. They can be
 10289 built from the encoder-decoder architecture discussed in § 18.3 and § 19.1.2: the encoder
 10290 converts the user’s input into a vector, and the decoder produces a sequence of words
 10291 as a response. For example, Shang et al. (2015) apply the attentional encoder-decoder
 10292 translation model, training on a dataset of posts and responses from the Chinese microblogging
 10293 platform Sina Weibo.⁵ This approach is capable of generating replies that relate thematically
 10294 to the input, as shown in the following examples.⁶

10295 (19.10) A: High fever attacks me every New Year’s day.
 10296 Get B: well soon and stay healthy!

10297 (19.11) A: I gain one more year. Grateful to my group, so happy.
 10298 B: Getting old now. Time has no mercy.

10299 While encoder-decoder models can generate responses that make sense in the context of
 10300 the immediately preceding turn, they struggle to maintain coherence over longer conversations.
 10301 One solution is to model the dialogue context recurrently. This creates a hierarchical
 10302 recurrent network, including both word-level and turn-level recurrences. The turn-level
 10303 hidden state is then used as additional context in the decoder (Serban et al., 2016), as
 10304 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).

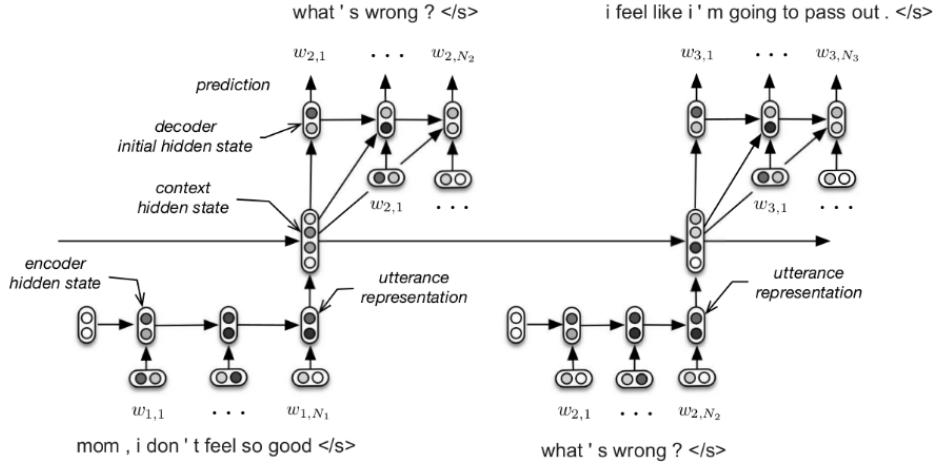


Figure 19.6: A hierarchical recurrent neural network for dialogue, with recurrence over both words and turns, from Serban et al. (2016). [todo: permission]

10305 An open question is how to integrate the encoder-decoder architecture into task-oriented
 10306 dialogue systems. Neural chatbots can be trained end-to-end: the user’s turn is analyzed
 10307 by the encoder, and the system output is generated by the decoder. This architecture can
 10308 be trained by log-likelihood using backpropagation (e.g., Sordoni et al., 2015; Serban et al.,
 10309 2016), or by more elaborate objectives, using reinforcement learning (Li et al., 2016). In
 10310 contrast, the task-oriented dialogue systems described in § 19.3.1 typically involve a set of
 10311 specialized modules: one for recognizing the user input, another for deciding what action
 10312 to take, and a third for arranging the text of the system output.

10313 Recurrent neural network decoders can be integrated into Markov Decision Process
 10314 dialogue systems, by conditioning the decoder on a representation of the information that
 10315 is to be expressed in each turn (Wen et al., 2015). Specifically, the long short-term memory
 10316 (LSTM; § 6.3) architecture is augmented so that the memory cell at turn m takes an
 10317 additional input d_m , which is a representation of the slots and values to be expressed in
 10318 the next turn. However, this approach still relies on additional modules to recognize the
 10319 user’s utterance and to plan the overall arc of the dialogue.

10320 Another promising direction is to create embeddings for the elements in the domain:
 10321 for example, the slots in a record and the entities that can fill them. The encoder then
 10322 encodes not only the words of the user’s input, but the embeddings of the elements that
 10323 the user mentions. Similarly, the decoder is endowed with the ability to refer to specific
 10324 elements in the knowledge base. He et al. (2017) show that such a method can learn to
 10325 play a collaborative dialogue game, in which both players are given a list of entities and

10326 their properties, and the goal is to find an entity that is on both players' lists.

10327 **Further reading**

10328 Gatt and Krahmer (2018) provide a comprehensive recent survey on text generation. For
10329 a book-length treatment of earlier work, see Reiter and Dale (2000). For a survey on
10330 image captioning, see Bernardi et al. (2016); for a survey of pre-neural approaches to
10331 dialogue systems, see Rieser and Lemon (2011). Dialogue acts were introduced in § 8.6
10332 as a labeling scheme for human-human dialogues; they also play a critical role in task-based
10333 dialogue systems (e.g., Allen et al., 1996). The incorporation of theoretical models of
10334 dialogue into computational systems is reviewed by Jurafsky and Martin (2009, chapter
10335 24).

10336 While this chapter has focused on the informative dimension of text generation, another
10337 line of research aims to generate text with configurable stylistic properties (Walker et al.,
10338 1997; Mairesse and Walker, 2011; Ficler and Goldberg, 2017; Hu et al., 2017). This chapter
10339 also does not address the generation of creative text such as narratives (Riedl and Young,
10340 2010), jokes (Ritchie, 2001), poems (Colton et al., 2012), and song lyrics (Gonçalo Oliveira
10341 et al., 2007).

10342 **Exercises**

10343 1. The SimpleNLG system produces surface realizations from representations of desired
10344 syntactic structure (Gatt and Reiter, 2009). This system can be accessed on github
10345 at <https://github.com/simplenlg/simplenlg>. Download the system, and produce
10346 realizations of the following examples:

- 10347 (19.12) Call me Ismael.
10348 (19.13) I try all things.
10349 (19.14) I achieve what I can.

10350 Then convert each example to a question. [todo: Can't get SimpleNLG to work with
10351 python anymore]

10352 Appendix A

10353 Probability

10354 Probability theory provides a way to reason about random events. The sorts of random
10355 events that are typically used to explain probability theory include coin flips, card draws,
10356 and the weather. It may seem odd to think about the choice of a word as akin to the flip of
10357 a coin, particularly if you are the type of person to choose words carefully. But random or
10358 not, language has proven to be extremely difficult to model deterministically. Probability
10359 offers a powerful tool for modeling and manipulating linguistic data.

10360 Probability can be thought of in terms of random outcomes: for example, a single coin
10361 flip has two possible outcomes, heads or tails. The set of possible outcomes is the sample
10362 space, and a subset of the sample space is an event. For a sequence of two coin flips,
10363 there are four possible outcomes, $\{HH, HT, TH, TT\}$, representing the ordered sequences
10364 heads-head, heads-tails, tails-heads, and tails-tails. The event of getting exactly one head
10365 includes two outcomes: $\{HT, TH\}$.

10366 Formally, a probability is a function from events to the interval between zero and
10367 one: $\Pr : \mathcal{F} \mapsto [0, 1]$, where \mathcal{F} is the set of possible events. An event that is certain
10368 has probability one; an event that is impossible has probability zero. For example, the
10369 probability of getting fewer than three heads on two coin flips is one. Each outcome is also
10370 an event (a set with exactly one element), and for two flips of a fair coin, the probability
10371 of each outcome is,

$$\Pr(\{HH\}) = \Pr(\{HT\}) = \Pr(\{TH\}) = \Pr(\{TT\}) = \frac{1}{4}. \quad [\text{A.1}]$$

10372 A.1 Probabilities of event combinations

10373 Because events are sets of outcomes, we can use set-theoretic operations such as complement,
10374 intersection, and union to reason about the probabilities of events and their combinations.

10375 For any event A , there is a complement $\neg A$, such that:

- 10376 • The probability of the union $A \cup \neg A$ is $\Pr(A \cup \neg A) = 1$;
 10377 • The intersection $A \cap \neg A = \emptyset$ is the empty set, and $\Pr(A \cap \neg A) = 0$.

10378 In the coin flip example, the event of obtaining a single head on two flips corresponds to
 10379 the set of outcomes $\{HT, TH\}$; the complement event includes the other two outcomes,
 10380 $\{TT, HH\}$.

10381 A.1.1 Probabilities of disjoint events

10382 When two events have an empty intersection, $A \cap B = \emptyset$, they are disjoint. The probability
 10383 of the union of two disjoint events is equal to the sum of their probabilities,

$$A \cap B = \emptyset \Rightarrow \Pr(A \cup B) = \Pr(A) + \Pr(B). \quad [A.2]$$

10384 This is the third axiom of probability, and it can be generalized to any countable sequence
 10385 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,

$$\Pr(\{HT, TH\}) = \Pr(\{HT\} \cup \{TH\}) = \Pr(\{HT\}) + \Pr(\{TH\}) \quad [A.3]$$

$$= \frac{1}{4} + \frac{1}{4} = \frac{1}{2}. \quad [A.4]$$

10386 In the general, the probability of the union of two events is,

$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B). \quad [A.5]$$

This can be seen visually in Figure A.1, and it can be derived from the third axiom of probability. Consider an event that includes all outcomes in B that are not in A , denoted as $B - (A \cap B)$. By construction, this event is disjoint from A . We can therefore apply the additive rule,

$$\Pr(A \cup B) = \Pr(A) + \Pr(B - (A \cap B)). \quad [A.6]$$

Furthermore, the event B is the union of two disjoint events: $A \cap B$ and $B - (A \cap B)$.

$$\Pr(B) = \Pr(B - (A \cap B)) + \Pr(A \cap B). \quad [A.7]$$

Reorganizing and substituting into Equation A.6 gives the desired result:

$$\Pr(B - (A \cap B)) = \Pr(B) - \Pr(A \cap B) \quad [A.8]$$

$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B). \quad [A.9]$$

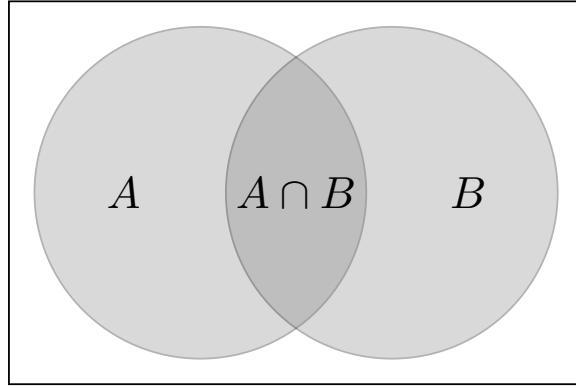


Figure A.1: A visualization of the probability of non-disjoint events A and B .

10387 A.1.2 Law of total probability

10388 A set of events $\mathcal{B} = \{B_1, B_2, \dots, B_N\}$ is a partition of the sample space iff each pair of
10389 events is disjoint ($B_i \cap B_j = \emptyset$), and the union of the events is the entire sample space.
10390 The law of total probability states that we can marginalize over these events as follows,

$$\Pr(A) = \sum_{B_n \in \mathcal{B}} \Pr(A \cap B_n). \quad [\text{A.10}]$$

10391 For any event B , the union $B \cup \neg B$ is a partition of the sample space. Therefore, a special
10392 case of the law of total probability is,

$$\Pr(A) = \Pr(A \cap B) + \Pr(A \cap \neg B). \quad [\text{A.11}]$$

10393 A.2 Conditional probability and Bayes' rule

A conditional probability is an expression like $\Pr(A | B)$, 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,

$$\Pr(A | B) = \frac{\Pr(A \cap B)}{\Pr(B)}. \quad [\text{A.12}]$$

The chain rule of probability states that $\Pr(A \cap B) = \Pr(A | B) \times \Pr(B)$, which is just a rearrangement of terms from Equation A.12. The chain rule can be applied repeatedly:

$$\begin{aligned} \Pr(A \cap B \cap C) &= \Pr(A | B \cap C) \times \Pr(B \cap C) \\ &= \Pr(A | B \cap C) \times \Pr(B | C) \times \Pr(C). \end{aligned}$$

Bayes' rule (sometimes called Bayes' law or Bayes' theorem) gives us a way to convert between $\Pr(A | B)$ and $\Pr(B | A)$. It follows from the definition of conditional probability and the chain rule:

$$\Pr(A | B) = \frac{\Pr(A \cap B)}{\Pr(B)} = \frac{\Pr(B | A) \times \Pr(A)}{\Pr(B)} \quad [A.13]$$

10394 Each term in Bayes rule has a name, which we will occasionally use:

- 10395 • $\Pr(A)$ is the prior, since it is the probability of event A without knowledge about
10396 whether B happens or not.
- 10397 • $\Pr(B | A)$ is the likelihood, the probability of event B given that event A has occurred.
- 10398 • $\Pr(A | B)$ is the posterior, the probability of event A with knowledge that B has
10399 occurred.

10400 Example The classic examples for Bayes' rule involve tests for rare diseases, but Manning
10401 and Schütze (1999) reframe this example in a linguistic setting. Suppose that you are interested
10402 in a rare syntactic construction, such as parasitic gaps, which occur on average
10403 once in 100,000 sentences. Here is an example of a parasitic gap:

10404 (A.1) Which class did you attend ___ without registering for ___?

10405 Lana Linguist has developed a complicated pattern matcher that attempts to identify
10406 sentences with parasitic gaps. It's pretty good, but it's not perfect:

- 10407 • If a sentence has a parasitic gap, the pattern matcher will find it with probability
10408 0.95. (This is the recall, which is one minus the false positive rate.)
- 10409 • If the sentence doesn't have a parasitic gap, the pattern matcher will wrongly say it
10410 does with probability 0.005. (This is the false positive rate, which is one minus the
10411 precision.)

10412 Suppose that Lana's pattern matcher says that a sentence contains a parasitic gap. What
10413 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:

$$\Pr(G | T) = \frac{\Pr(T | G) \times \Pr(G)}{\Pr(T)}. \quad [A.14]$$

10414 We already know both terms in the numerator: $\Pr(T | G)$ is the recall, which is 0.95; $\Pr(G)$
 10415 is the prior, which is 10^{-5} .

10416 We are not given the denominator, but it can be computed using tools developed earlier
 10417 in this section. First apply the law of total probability, using the partition $\{G, \neg G\}$:

$$\Pr(T) = \Pr(T \cap G) + \Pr(T \cap \neg G). \quad [\text{A.15}]$$

This says that the probability of the test being positive is the sum of the probability of a true positive ($T \cap G$) and the probability of a false positive ($T \cap \neg G$). The probability of each of these events can be computed using the chain rule:

$$\Pr(T \cap G) = \Pr(T | G) \times \Pr(G) = 0.95 \times 10^{-5} \quad [\text{A.16}]$$

$$\Pr(T \cap \neg G) = \Pr(T | \neg G) \times \Pr(\neg G) = 0.005 \times (1 - 10^{-5}) \approx 0.005 \quad [\text{A.17}]$$

$$\Pr(T) = \Pr(T \cap G) + \Pr(T \cap \neg G) \quad [\text{A.18}]$$

$$= 0.95 \times 10^{-5} + 0.005. \quad [\text{A.19}]$$

Plugging these terms into Bayes' rule gives the desired posterior probability,

$$\Pr(G | T) = \frac{\Pr(T | G) \Pr(G)}{\Pr(T)} \quad [\text{A.20}]$$

$$= \frac{0.95 \times 10^{-5}}{0.95 \times 10^{-5} + 0.005 \times (1 - 10^{-5})} \quad [\text{A.21}]$$

$$\approx 0.002. \quad [\text{A.22}]$$

10418 Lana's pattern matcher seems accurate, with false positive and false negative rates
 10419 below 5%. Yet the extreme rarity of the phenomenon means that a positive result from
 10420 the detector is most likely to be wrong.

10421 A.3 Independence

Two events are independent if the probability of their intersection is equal to the product of their probabilities: $\Pr(A \cap B) = \Pr(A) \times \Pr(B)$. For example, for two flips of a fair coin, the probability of getting heads on the first flip is independent of the probability of getting

heads on the second flip:

$$\Pr(\{HT, HH\}) = \Pr(HT) + \Pr(HH) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2} \quad [A.23]$$

$$\Pr(\{HH, TH\}) = \Pr(HH) + \Pr(TH) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2} \quad [A.24]$$

$$\Pr(\{HT, HH\}) \times \Pr(\{HH, TH\}) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4} \quad [A.25]$$

$$\Pr(\{HT, HH\} \cap \{HH, TH\}) = \Pr(HH) = \frac{1}{4} \quad [A.26]$$

$$= \Pr(\{HT, HH\}) \times \Pr(\{HH, TH\}). \quad [A.27]$$

If $\Pr(A \cap B \mid C) = \Pr(A \mid C) \times \Pr(B \mid C)$, then the events A and B are conditionally independent, written $A \perp B \mid C$. Conditional independence plays a important role in probabilistic models such as Naïve Bayes chapter 2.

A.4 Random variables

Random variables are functions from events to \mathbb{R}^n , where \mathbb{R} is the set of real numbers. This subsumes several useful special cases:

- An indicator random variable is a function from events to the set $\{0, 1\}$. In the coin flip example, we can define Y as an indicator random variable, taking the value 1 when the coin has come up heads on at least one flip. This would include the outcomes $\{HH, HT, TH\}$. The probability $\Pr(Y = 1)$ is the sum of the probabilities of these outcomes, $\Pr(Y = 1) = \frac{1}{4} + \frac{1}{4} + \frac{1}{4} = \frac{3}{4}$.
- A discrete random variable is a function from events to a discrete subset of \mathbb{R} . Consider the coin flip example: the number of heads on two flips, X , can be viewed as a discrete random variable, $X \in \{0, 1, 2\}$. The event probability $\Pr(X = 1)$ can again be computed as the sum of the probabilities of the events in which there is one head, $\{HT, TH\}$, giving $\Pr(X = 1) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$.

Each possible value of a random variable is associated with a subset of the sample space. In the coin flip example, $X = 0$ is associated with the event $\{TT\}$, $X = 1$ is associated with the event $\{HT, TH\}$, and $X = 2$ is associated with the event $\{HH\}$. Assuming a fair coin, the probabilities of these events are, respectively, $1/4$, $1/2$, and $1/4$. This list of numbers represents the probability distribution over X , written p_X , which maps from the possible values of X to the non-negative reals. For a specific value x , we write $p_X(x)$, which is equal to the event probability $\Pr(X = x)$.¹ The function p_X is called a probability mass function

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 $p(x)$.

(pmf) if X is discrete; it is called a probability density function (pdf) if X is continuous. In either case, the function must sum to one, and all values must be non-negative:

$$\int_x p_X(x)dx = 1 \quad [A.28]$$

$$\forall x, p_X(x) \geq 0. \quad [A.29]$$

10438 Probabilities over multiple random variables can written as joint probabilities, e.g.,
 10439 $p_{A,B}(a,b) = \Pr(A = a \cap B = b)$. Several properties of event probabilities carry over to
 10440 probability distributions over random variables:

- 10441 • The marginal probability distribution is $p_A(a) = \sum_b p_{A,B}(a,b)$.
- 10442 • The conditional probability distribution is $p_{A|B}(a | b) = \frac{p_{A,B}(a,b)}{p_B(b)}$.
- 10443 • Random variables A and B are independent iff $p_{A,B}(a,b) = p_A(a) \times p_B(b)$.

10444 A.5 Expectations

10445 Sometimes we want the expectation of a function, such as $E[g(x)] = \sum_{x \in \mathcal{X}} g(x)p(x)$.
 10446 Expectations are easiest to think about in terms of probability distributions over discrete
 10447 events:

- 10448 • If it is sunny, Lucia will eat three ice creams.
- 10449 • If it is rainy, she will eat only one ice cream.
- 10450 • There's a 80% chance it will be sunny.
- 10451 • The expected number of ice creams she will eat is $0.8 \times 3 + 0.2 \times 1 = 2.6$.

10452 If the random variable X is continuous, the expectation is an integral:

$$E[g(x)] = \int_{\mathcal{X}} g(x)p(x)dx \quad [A.30]$$

10453 For example, a fast food restaurant in Quebec has a special offer for cold days: they give a
 10454 1% discount on poutine for every degree below zero. Assuming a thermometer with infinite
 10455 precision, the expected price would be an integral over all possible temperatures,

$$E[\text{price}(x)] = \int_{\mathcal{X}} \min(1, 1+x) \times \text{original-price} \times p(x)dx. \quad [A.31]$$

10456 A.6 Modeling and estimation

10457 Probabilistic models provide a principled way to reason about random events and random
10458 variables. Let's consider the coin toss example. Each toss can be modeled as a random
10459 event, with probability θ of the event H , and probability $1 - \theta$ of the complementary event
10460 T . If we write a random variable X as the total number of heads on three coin flips,
10461 then the distribution of X depends on θ . In this case, X is distributed as a binomial
10462 random variable, meaning that it is drawn from a binomial distribution, with parameters
10463 $(\theta, N = 3)$. This is written,

$$X \sim \text{Binomial}(\theta, N = 3). \quad [\text{A.32}]$$

10464 The properties of the binomial distribution enable us to make statements about the X ,
10465 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 executed N trials, and obtained x heads. We can estimate θ by the principle of maximum likelihood:

$$\hat{\theta} = \operatorname{argmax}_{\theta} p_X(x; \theta, N). \quad [\text{A.33}]$$

This says that the estimate $\hat{\theta}$ should be the value that maximizes the likelihood of the data. The semicolon indicates that θ and N are parameters of the probability function. The likelihood $p_X(x; \theta, N)$ can be computed from the binomial distribution,

$$p_X(x; \theta, N) = \frac{N!}{x!(N-x)!} \theta^x (1 - \theta)^{N-x}. \quad [\text{A.34}]$$

10466 This likelihood is proportional to the product of the probability of individual outcomes:
10467 for example, the sequence T, H, H, T, H would have probability $\theta^3(1 - \theta)^2$. The term
10468 $\frac{N!}{x!(N-x)!}$ arises from the many possible orderings by which we could obtain x heads on N
10469 trials. This term does not depend on θ , so it can be ignored during estimation.

In practice, we maximize the log-likelihood, which is a monotonic function of the 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.

$$\ell(\theta) = x \log \theta + (N - x) \log(1 - \theta) \quad [\text{A.35}]$$

$$\frac{\partial \ell(\theta)}{\partial \theta} = \frac{x}{\theta} - \frac{N - x}{1 - \theta} \quad [\text{A.36}]$$

$$\frac{N - x}{1 - \theta} = \frac{x}{\theta} \quad [\text{A.37}]$$

$$\frac{N - x}{x} = \frac{1 - \theta}{\theta} \quad [\text{A.38}]$$

$$\frac{N}{x} - 1 = \frac{1}{\theta} - 1 \quad [\text{A.39}]$$

$$\hat{\theta} = \frac{x}{N}. \quad [\text{A.40}]$$

10470 In this case, the maximum likelihood estimate is equal to $\frac{x}{N}$, the fraction of trials that
 10471 came up heads. This intuitive solution is also known as the relative frequency estimate,
 10472 since it is equal to the relative frequency of the outcome.

Is maximum likelihood estimation always the right choice? Suppose you conduct one trial, and get heads. Would you conclude that $\theta = 1$, meaning that the coin is guaranteed to come up heads? If not, then you must have some prior expectation about θ . To incorporate this prior information, we can treat θ as a random variable, and use Bayes' rule:

$$p(\theta | x; N) = \frac{p(x | \theta) \times p(\theta)}{p(x)} \quad [\text{A.41}]$$

$$\propto p(x | \theta) \times p(\theta) \quad [\text{A.42}]$$

$$\hat{\theta} = \operatorname{argmax}_{\theta} p(x | \theta) \times p(\theta). \quad [\text{A.43}]$$

10473 This is the maximum a posteriori (MAP) estimate. Given a form for $p(\theta)$, you can derive
 10474 the MAP estimate using the same approach that was used to derive the maximum likelihood
 10475 estimate.

10476 Additional resources

10477 A good introduction to probability theory is offered by Manning and Schütze (1999), which
 10478 helped to motivate this section. For more detail, Sharon Goldwater provides another
 10479 useful reference, <http://homepages.inf.ed.ac.uk/sgwater/teaching/general/probability.pdf>.
 10480 A historical and philosophical perspective on probability is offered by Diaconis and Skyrms
 10481 (2017).

₁₀₄₈₂ Appendix B

₁₀₄₈₃ Numerical optimization

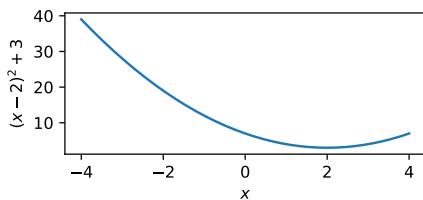
₁₀₄₈₄ Unconstrained numerical optimization involves solving problems of the form,

$$\min_{\mathbf{x} \in \mathbb{R}^D} f(\mathbf{x}), \quad [\text{B.1}]$$

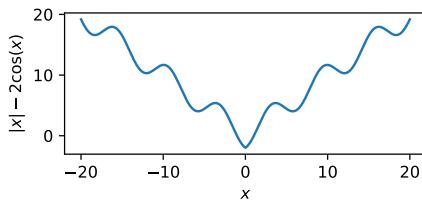
₁₀₄₈₅ where $\mathbf{x} \in \mathbb{R}^D$ is a vector of D real numbers.

₁₀₄₈₆ Differentiation is fundamental to continuous optimization. Suppose that at some \mathbf{x}^* ,
₁₀₄₈₇ 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
₁₀₄₈₈ point of f . For a convex function f (defined in § 2.3), $f(\mathbf{x}^*)$ is equal to the global minimum
₁₀₄₈₉ of f iff \mathbf{x}^* is a critical point of f .

As an example, consider the convex function $f(x) = (x - 2)^2 + 3$, shown in Figure B.1a. The derivative is $\frac{\partial f}{\partial x} = 2x - 4$. A unique minimum can be obtained by setting the derivative equal to zero and solving for x , obtaining $x^* = 2$. Now consider the multivariate convex function $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{x} - [2, 1]^\top\|^2$, where $\|\mathbf{x}\|^2$ is the squared Euclidean norm. The partial



(a) The function $f(x) = (x - 2)^2 + 3$



(b) The function $f(x) = |x| - 2\cos(x)$

Figure B.1: Two functions with unique global minima

derivatives are,

$$\frac{\partial d}{\partial x_1} = x_1 - 2 \quad [B.2]$$

$$\frac{\partial d}{\partial x_2} = x_2 - 1 \quad [B.3]$$

10490 The unique minimum is $\mathbf{x}^* = [2, 1]^\top$.

10491 For non-convex functions, critical points are not necessarily global minima. A local
 10492 minimum \mathbf{x}^* is a point at which the function takes a smaller value than at all nearby
 10493 neighbors: formally, \mathbf{x}^* is a local minimum if there is some positive ϵ such that $f(\mathbf{x}^*) \leq f(\mathbf{x})$
 10494 for all \mathbf{x} within distance ϵ of \mathbf{x}^* . Figure B.1b shows the function $f(x) = |x| - 2 \cos(x)$,
 10495 which has many local minima, as well as a unique global minimum at $x = 0$. A critical
 10496 point may also be the local or global maximum of the function; it may be a saddle point,
 10497 which is a minimum with respect to at least one coordinate, and a maximum with respect
 10498 to at least one other coordinate; it may be an inflection point, which is neither a minimum
 10499 nor maximum. When available, the second derivative of f can help to distinguish these
 10500 cases.

10501 B.1 Gradient descent

For many convex functions, it is not possible to solve for \mathbf{x}^* in closed form. In gradient descent, we compute a series of solutions, $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \dots$ by taking steps along the local gradient $\nabla_{\mathbf{x}^{(t)}} f$, which is the vector of partial derivatives of the function f , evaluated at the point $\mathbf{x}^{(t)}$. Each solution $\mathbf{x}^{(t+1)}$ is computed,

$$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} - \eta^{(t)} \nabla_{\mathbf{x}^{(t)}} f. \quad [B.4]$$

10502 where $\eta^{(t)} > 0$ is a step size. If the step size is chosen appropriately, this procedure will find
 10503 the global minimum of a differentiable convex function. For non-convex functions, gradient
 10504 descent will find a local minimum. The extension to non-differentiable convex functions is
 10505 discussed in § 2.3.

10506 B.2 Constrained optimization

Optimization must often be performed under constraints: for example, when optimizing the parameters of a probability distribution, the probabilities of all events must sum to one. Constrained optimization problems can be written,

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad [B.5]$$

$$\text{s.t. } g_c(\mathbf{x}) \leq 0, \quad \forall c = 1, 2, \dots, C \quad [B.6]$$

where each $g_i(\mathbf{x})$ is a scalar function of \mathbf{x} . For example, suppose that \mathbf{x} must be non-negative, and that its sum cannot exceed a budget b . Then there are $D + 1$ inequality constraints,

$$g_i(\mathbf{x}) = -x_i, \quad \forall i = 1, 2, \dots, D \quad [\text{B.7}]$$

$$g_{D+1}(\mathbf{x}) = -b + \sum_{i=1}^D x_i. \quad [\text{B.8}]$$

Inequality constraints can be combined with the original objective function f by forming a Lagrangian,

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{c=1}^C \lambda_c g_c(\mathbf{x}), \quad [\text{B.9}]$$

where λ_c is a Lagrange multiplier. For any Lagrangian, there is a corresponding dual form, which is a function of $\boldsymbol{\lambda}$:

$$D(\boldsymbol{\lambda}) = \min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}). \quad [\text{B.10}]$$

The Lagrangian L can be referred to as the primal form.

B.3 Example: Passive-aggressive online learning

Sometimes it is possible to solve a constrained optimization problem by manipulating the Lagrangian. One example is maximum-likelihood estimation of a Naïve Bayes probability model, as described in § 2.1.3. In that case, it is unnecessary to explicitly compute the Lagrange multiplier. Another example is illustrated by the passive-aggressive algorithm for online learning (Crammer et al., 2006). This algorithm is similar to the perceptron, but the goal at each step is to make the most conservative update that gives zero margin loss on the current example.¹ Each update can be formulated as a constrained optimization over the weights $\boldsymbol{\theta}$:

$$\min_{\boldsymbol{\theta}} \frac{1}{2} \|\boldsymbol{\theta} - \boldsymbol{\theta}^{(i-1)}\|^2 \quad [\text{B.11}]$$

$$\text{s.t. } \ell^{(i)}(\boldsymbol{\theta}) = 0 \quad [\text{B.12}]$$

where $\boldsymbol{\theta}^{(i-1)}$ is the previous set of weights, and $\ell^{(i)}(\boldsymbol{\theta})$ is the margin loss on instance i . As in § 2.3.1, this loss is defined as,

$$\ell^{(i)}(\boldsymbol{\theta}) = 1 - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) + \max_{y \neq y^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y). \quad [\text{B.13}]$$

¹This is the basis for the name of the algorithm: it is passive when the loss is zero, but it aggressively moves to make the loss zero when necessary.

10515 When the margin loss is zero for $\boldsymbol{\theta}^{(i-1)}$, the optimal solution is simply to set $\boldsymbol{\theta}^* = \boldsymbol{\theta}^{(i-1)}$,
 10516 so we will focus on the case where $\ell^{(i)}(\boldsymbol{\theta}^{(i-1)}) > 0$. 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,

$$\nabla_{\boldsymbol{\theta}} L = \boldsymbol{\theta} - \boldsymbol{\theta}^{(i-1)} + \lambda \frac{\partial}{\partial \boldsymbol{\theta}} \ell^{(i)}(\boldsymbol{\theta}) \quad [\text{B.15}]$$

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}^{(i-1)} + \lambda \boldsymbol{\delta}, \quad [\text{B.16}]$$

10517 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-style update to $\boldsymbol{\theta}$. We can solve for λ 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 ,

$$\frac{\partial D}{\partial \lambda} = -\lambda \|\boldsymbol{\delta}\|^2 + \ell^{(i)}(\boldsymbol{\theta}^{(i-1)}) \quad [\text{B.20}]$$

$$\lambda^* = \frac{\ell^{(i)}(\boldsymbol{\theta}^{(i-1)})}{\|\boldsymbol{\delta}\|^2}. \quad [\text{B.21}]$$

10518 The complete update equation is therefore:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}^{(i-1)} + \frac{\ell^{(i)}(\boldsymbol{\theta}^{(i-1)})}{\|\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - \mathbf{f}(\mathbf{x}^{(i)}, \hat{y})\|^2} (\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - \mathbf{f}(\mathbf{x}^{(i)}, \hat{y})). \quad [\text{B.22}]$$

10519 This update has strong intuitive support. The numerator of the learning rate grows with
 10520 the loss. The denominator grows with the norm of the difference between the feature
 10521 vectors associated with the correct and predicted label. If this norm is large, then the step
 10522 with respect to each feature should be small, and vice versa.

¹⁰⁵²³ Bibliography

- 10524 Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis,
10525 J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia,
10526 R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G.
10527 Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker,
10528 V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg,
10529 M. Wicke, Y. Yu, and X. Zheng (2016). Tensorflow: Large-scale machine learning on
10530 heterogeneous distributed systems. CoRR abs/1603.04467.
- 10531 Abend, O. and A. Rappoport (2017). The state of the art in semantic representation. In
10532 Proceedings of the Association for Computational Linguistics (ACL).
- 10533 Abney, S., R. E. Schapire, and Y. Singer (1999). Boosting applied to tagging and PP
10534 attachment. In Proceedings of Empirical Methods for Natural Language Processing
10535 (EMNLP), pp. 132–134.
- 10536 Abney, S. P. (1987). The English noun phrase in its sentential aspect. Ph. D. thesis,
10537 Massachusetts Institute of Technology.
- 10538 Abney, S. P. and M. Johnson (1991). Memory requirements and local ambiguities of parsing
10539 strategies. Journal of Psycholinguistic Research 20(3), 233–250.
- 10540 Adafre, S. F. and M. De Rijke (2006). Finding similar sentences across multiple languages
10541 in wikipedia. In Proceedings of the Workshop on NEW TEXT Wikis and blogs and
10542 other dynamic text sources.
- 10543 Ahn, D. (2006). The stages of event extraction. In Proceedings of the Workshop
10544 on Annotating and Reasoning about Time and Events, pp. 1–8. Association for
10545 Computational Linguistics.
- 10546 Aho, A. V., M. S. Lam, R. Sethi, and J. D. Ullman (2006). Compilers: Principles,
10547 techniques, & tools.
- 10548 Aikhenvald, A. Y. (2004). Evidentiality. Oxford University Press.

- 10549 Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions
10550 on Automatic Control* 19(6), 716–723.
- 10551 Akmajian, A., R. A. Demers, A. K. Farmer, and R. M. Harnish (2010). *Linguistics: An
10552 introduction to language and communication* (Sixth ed.). Cambridge, MA: MIT press.
- 10553 Alfauf, F. (1999). Chromos. Dalkey Archive Press.
- 10554 Allauzen, C., M. Riley, J. Schalkwyk, W. Skut, and M. Mohri (2007). OpenFst: A general
10555 and efficient weighted finite-state transducer library. In *International Conference on
10556 Implementation and Application of Automata*, pp. 11–23. Springer.
- 10557 Allen, J. F. (1984). Towards a general theory of action and time. *Artificial intelligence* 23(2),
10558 123–154.
- 10559 Allen, J. F., B. W. Miller, E. K. Ringger, and T. Sikorski (1996). A robust system for
10560 natural spoken dialogue. In *Proceedings of the Association for Computational Linguistics
10561 (ACL)*, pp. 62–70.
- 10562 Allen, J. F., L. K. Schubert, G. Ferguson, P. Heeman, C. H. Hwang, T. Kato, M. Light,
10563 N. Martin, B. Miller, M. Poesio, and D. Traum (1995). The TRAINS project: A case
10564 study in building a conversational planning agent. *Journal of Experimental & Theoretical
10565 Artificial Intelligence* 7(1), 7–48.
- 10566 Alm, C. O., D. Roth, and R. Sproat (2005). Emotions from text: machine learning
10567 for text-based emotion prediction. In *Proceedings of Empirical Methods for Natural
10568 Language Processing (EMNLP)*, pp. 579–586.
- 10569 Aluísio, S., J. Pelizzoni, A. Marchi, L. de Oliveira, R. Manenti, and V. Marquiafável (2003).
10570 An account of the challenge of tagging a reference corpus for Brazilian Portuguese.
10571 *Computational Processing of the Portuguese Language*, 194–194.
- 10572 Anand, P., M. Walker, R. Abbott, J. E. Fox Tree, R. Bowman, and M. Minor (2011).
10573 Cats rule and dogs drool!: Classifying stance in online debate. In *Proceedings of the
10574 2nd Workshop on Computational Approaches to Subjectivity and Sentiment Analysis,
10575 Portland, Oregon*, pp. 1–9. Association for Computational Linguistics.
- 10576 Anandkumar, A. and R. Ge (2016). Efficient approaches for escaping higher order saddle
10577 points in non-convex optimization. In *Proceedings of the Conference On Learning Theory
10578 (COLT)*, pp. 81–102.
- 10579 Anandkumar, A., R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky (2014). Tensor
10580 decompositions for learning latent variable models. *The Journal of Machine Learning
10581 Research* 15(1), 2773–2832.

- 10582 Ando, R. K. and T. Zhang (2005). A framework for learning predictive structures from
10583 multiple tasks and unlabeled data. *The Journal of Machine Learning Research* 6, 1817–
10584 1853.
- 10585 Andor, D., C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and
10586 M. Collins (2016). Globally normalized transition-based neural networks. In *Proceedings*
10587 of the Association for Computational Linguistics (ACL), pp. 2442–2452.
- 10588 Angeli, G., P. Liang, and D. Klein (2010). A simple domain-independent probabilistic
10589 approach to generation. In *Proceedings of Empirical Methods for Natural Language
10590 Processing* (EMNLP), pp. 502–512.
- 10591 Antol, S., A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh
10592 (2015). Vqa: Visual question answering. In *Proceedings of the International Conference
10593 on Computer Vision* (ICCV), pp. 2425–2433.
- 10594 Aronoff, M. (1976). Word formation in generative grammar. MIT Press.
- 10595 Arora, S. and B. Barak (2009). Computational complexity: a modern approach. Cambridge
10596 University Press.
- 10597 Arora, S., R. Ge, Y. Halpern, D. Mimno, A. Moitra, D. Sontag, Y. Wu, and M. Zhu (2013).
10598 A practical algorithm for topic modeling with provable guarantees. In *Proceedings of
10599 the International Conference on Machine Learning* (ICML), pp. 280–288.
- 10600 Arora, S., Y. Li, Y. Liang, T. Ma, and A. Risteski (2016). Linear algebraic structure of
10601 word senses, with applications to polysemy. arXiv preprint arXiv:1601.03764.
- 10602 Artstein, R. and M. Poesio (2008). Inter-coder agreement for computational linguistics.
10603 *Computational Linguistics* 34(4), 555–596.
- 10604 Artzi, Y. and L. Zettlemoyer (2013). Weakly supervised learning of semantic parsers for
10605 mapping instructions to actions. *Transactions of the Association for Computational
10606 Linguistics* 1, 49–62.
- 10607 Attardi, G. (2006). Experiments with a multilanguage non-projective dependency parser.
10608 In *Proceedings of the Conference on Natural Language Learning* (CoNLL), pp. 166–170.
- 10609 Auer, P. (2013). Code-switching in conversation: Language, interaction and identity.
10610 Routledge.
- 10611 Auer, S., C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives (2007). Dbpedia:
10612 A nucleus for a web of open data. *The semantic web*, 722–735.
- 10613 Austin, J. L. (1962). How to do things with words. Oxford University Press.

- 10614 Aw, A., M. Zhang, J. Xiao, and J. Su (2006). A phrase-based statistical model for SMS text
 10615 normalization. In Proceedings of the Association for Computational Linguistics (ACL),
 10616 pp. 33–40.
- 10617 Ba, J. L., J. R. Kiros, and G. E. Hinton (2016). Layer normalization. arXiv preprint
 10618 arXiv:1607.06450.
- 10619 Bagga, A. and B. Baldwin (1998a). Algorithms for scoring coreference chains. In
 10620 Proceedings of the Language Resources and Evaluation Conference, pp. 563–566.
- 10621 Bagga, A. and B. Baldwin (1998b). Entity-based cross-document coreferencing using the
 10622 vector space model. In Proceedings of the International Conference on Computational
 10623 Linguistics (COLING), pp. 79–85.
- 10624 Bahdanau, D., K. Cho, and Y. Bengio (2014). Neural machine translation by jointly
 10625 learning to align and translate. In Neural Information Processing Systems (NIPS).
- 10626 Baldwin, T. and S. N. Kim (2010). Multiword expressions. In Handbook of natural language
 10627 processing, Volume 2, pp. 267–292. Boca Raton, USA: CRC Press.
- 10628 Balle, B., A. Quattoni, and X. Carreras (2011). A spectral learning algorithm for finite
 10629 state transducers. In Proceedings of the European Conference on Machine Learning and
 10630 Principles and Practice of Knowledge Discovery in Databases (ECML), pp. 156–171.
- 10631 Banarescu, L., C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight,
 10632 P. Koehn, M. Palmer, and N. Schneider (2013, August). Abstract meaning
 10633 representation for sembanking. In Proceedings of the 7th Linguistic Annotation
 10634 Workshop and Interoperability with Discourse, Sofia, Bulgaria, pp. 178–186. Association
 10635 for Computational Linguistics.
- 10636 Banko, M., M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni (2007).
 10637 Open information extraction from the web. In Proceedings of the International Joint
 10638 Conference on Artificial Intelligence (IJCAI), pp. 2670–2676.
- 10639 Bansal, N., A. Blum, and S. Chawla (2004). Correlation clustering. Machine Learning 56(1–
 10640 3), 89–113.
- 10641 Barber, D. (2012). Bayesian reasoning and machine learning. Cambridge University Press.
- 10642 Barman, U., A. Das, J. Wagner, and J. Foster (2014, October). Code mixing: A challenge
 10643 for language identification in the language of social media. In Proceedings of the First
 10644 Workshop on Computational Approaches to Code Switching, Doha, Qatar, pp. 13–23.
 10645 Association for Computational Linguistics.

- 10646 Barnickel, T., J. Weston, R. Collobert, H.-W. Mewes, and V. Stümpflen (2009). Large
10647 scale application of neural network based semantic role labeling for automated relation
10648 extraction from biomedical texts. *PLoS One* 4(7), e6393.
- 10649 Baron, A. and P. Rayson (2008). Vard2: A tool for dealing with spelling variation in
10650 historical corpora. In Postgraduate conference in corpus linguistics.
- 10651 Baroni, M., R. Bernardi, and R. Zamparelli (2014). Frege in space: A program for
10652 compositional distributional semantics. *Linguistic Issues in Language Technologies*.
- 10653 Barzilay, R. and M. Lapata (2008, mar). Modeling local coherence: An Entity-Based
10654 approach. *Computational Linguistics* 34(1), 1–34.
- 10655 Barzilay, R. and K. R. McKeown (2005). Sentence fusion for multidocument news
10656 summarization. *Computational Linguistics* 31(3), 297–328.
- 10657 Beesley, K. R. and L. Karttunen (2003). Finite-state morphology. Stanford, CA: Center
10658 for the Study of Language and Information.
- 10659 Bejan, C. A. and S. Harabagiu (2014). Unsupervised event coreference resolution.
10660 *Computational Linguistics* 40(2), 311–347.
- 10661 Bell, E. T. (1934). Exponential numbers. *The American Mathematical Monthly* 41(7),
10662 411–419.
- 10663 Bender, E. M. (2013, jun). *Linguistic Fundamentals for Natural Language Processing:*
10664 100 Essentials from Morphology and Syntax, Volume 6 of *Synthesis Lectures on Human*
10665 *Language Technologies*. Morgan & Claypool Publishers.
- 10666 Bengio, S., O. Vinyals, N. Jaitly, and N. Shazeer (2015). Scheduled sampling for sequence
10667 prediction with recurrent neural networks. In *Neural Information Processing Systems*
10668 (NIPS), pp. 1171–1179.
- 10669 Bengio, Y., R. Ducharme, P. Vincent, and C. Janvin (2003). A neural probabilistic language
10670 model. *The Journal of Machine Learning Research* 3, 1137–1155.
- 10671 Bengio, Y., P. Simard, and P. Frasconi (1994). Learning long-term dependencies with
10672 gradient descent is difficult. *IEEE Transactions on Neural Networks* 5(2), 157–166.
- 10673 Bengtson, E. and D. Roth (2008). Understanding the value of features for coreference
10674 resolution. In *Proceedings of Empirical Methods for Natural Language Processing*
10675 (EMNLP), pp. 294–303.
- 10676 Benjamini, Y. and Y. Hochberg (1995). Controlling the false discovery rate: a practical and
10677 powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B*
10678 (Methodological), 289–300.

- 10679 Berant, J., A. Chou, R. Frostig, and P. Liang (2013). Semantic parsing on freebase
10680 from question-answer pairs. In Proceedings of Empirical Methods for Natural Language
10681 Processing (EMNLP), pp. 1533–1544.
- 10682 Berant, J., V. Srikumar, P.-C. Chen, A. Vander Linden, B. Harding, B. Huang, P. Clark,
10683 and C. D. Manning (2014). Modeling biological processes for reading comprehension. In
10684 Proceedings of Empirical Methods for Natural Language Processing (EMNLP).
- 10685 Berg-Kirkpatrick, T., A. Bouchard-Côté, J. DeNero, and D. Klein (2010). Painless
10686 unsupervised learning with features. In Proceedings of the North American Chapter
10687 of the Association for Computational Linguistics (NAACL), pp. 582–590.
- 10688 Berg-Kirkpatrick, T., D. Burkett, and D. Klein (2012). An empirical investigation of
10689 statistical significance in NLP. In Proceedings of Empirical Methods for Natural
10690 Language Processing (EMNLP), pp. 995–1005.
- 10691 Berger, A. L., V. J. D. Pietra, and S. A. D. Pietra (1996). A maximum entropy approach
10692 to natural language processing. *Computational linguistics* 22(1), 39–71.
- 10693 Bergsma, S., D. Lin, and R. Goebel (2008). Distributional identification of non-referential
10694 pronouns. In Proceedings of the Association for Computational Linguistics (ACL), pp.
10695 10–18.
- 10696 Bernardi, R., R. Cakici, D. Elliott, A. Erdem, E. Erdem, N. Ikizler-Cinbis, F. Keller,
10697 A. Muscat, and B. Plank (2016). Automatic description generation from images: A
10698 survey of models, datasets, and evaluation measures. *Journal of Artificial Intelligence
10699 Research* 55, 409–442.
- 10700 Bertsekas, D. P. (2012). Incremental gradient, subgradient, and proximal methods for
10701 convex optimization: A survey. See Sra et al. (2012).
- 10702 Bhatia, P., R. Guthrie, and J. Eisenstein (2016). Morphological priors for probabilistic
10703 neural word embeddings. In Proceedings of Empirical Methods for Natural Language
10704 Processing (EMNLP).
- 10705 Bhatia, P., Y. Ji, and J. Eisenstein (2015). Better document-level sentiment analysis
10706 from rst discourse parsing. In Proceedings of Empirical Methods for Natural Language
10707 Processing (EMNLP).
- 10708 Biber, D. (1991). Variation across speech and writing. Cambridge University Press.
- 10709 Bird, S., E. Klein, and E. Loper (2009). Natural language processing with Python.
10710 California: O'Reilly Media.
- 10711 Bishop, C. M. (2006). Pattern recognition and machine learning. Springer.

- 10712 Björkelund, A. and P. Nugues (2011). Exploring lexicalized features for coreference
10713 resolution. In Proceedings of the Conference on Natural Language Learning (CoNLL),
10714 pp. 45–50.
- 10715 Blackburn, P. and J. Bos (2005). Representation and inference for natural language: A
10716 first course in computational semantics. CSLI.
- 10717 Blei, D. M. (2012). Probabilistic topic models. Communications of the ACM 55(4), 77–84.
- 10718 Blei, D. M. (2014). Build, compute, critique, repeat: Data analysis with latent variable
10719 models. Annual Review of Statistics and Its Application 1, 203–232.
- 10720 Blei, D. M., A. Y. Ng, and M. I. Jordan (2003). Latent dirichlet allocation. the Journal of
10721 machine Learning research 3, 993–1022.
- 10722 Blitzer, J., M. Dredze, and F. Pereira (2007). Biographies, bollywood, boom-boxes
10723 and blenders: Domain adaptation for sentiment classification. In Proceedings of the
10724 Association for Computational Linguistics (ACL), pp. 440–447.
- 10725 Blum, A. and T. Mitchell (1998). Combining labeled and unlabeled data with co-training.
10726 In Proceedings of the Conference On Learning Theory (COLT), pp. 92–100.
- 10727 Bobrow, D. G., R. M. Kaplan, M. Kay, D. A. Norman, H. Thompson, and T. Winograd
10728 (1977). Gus, a frame-driven dialog system. Artificial intelligence 8(2), 155–173.
- 10729 Bohnet, B. (2010). Very high accuracy and fast dependency parsing is not a contradiction.
10730 In Proceedings of the International Conference on Computational Linguistics (COLING),
10731 pp. 89–97.
- 10732 Boitet, C. (1988). Pros and cons of the pivot and transfer approaches in multilingual
10733 machine translation. Readings in machine translation, 273–279.
- 10734 Bojanowski, P., E. Grave, A. Joulin, and T. Mikolov (2017). Enriching word vectors with
10735 subword information. Transactions of the Association for Computational Linguistics 5,
10736 135–146.
- 10737 Bollacker, K., C. Evans, P. Paritosh, T. Sturge, and J. Taylor (2008). Freebase: a
10738 collaboratively created graph database for structuring human knowledge. In Proceedings
10739 of the ACM International Conference on Management of Data (SIGMOD), pp. 1247–
10740 1250. ACM.
- 10741 Bolukbasi, T., K.-W. Chang, J. Y. Zou, V. Saligrama, and A. T. Kalai (2016). Man is
10742 to computer programmer as woman is to homemaker? debiasing word embeddings. In
10743 Neural Information Processing Systems (NIPS), pp. 4349–4357.

- 10744 Bordes, A., N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko (2013). Translating
 10745 embeddings for modeling multi-relational data. In Neural Information Processing
 10746 Systems (NIPS), pp. 2787–2795.
- 10747 Bordes, A., J. Weston, R. Collobert, Y. Bengio, et al. (2011). Learning structured
 10748 embeddings of knowledge bases. In Proceedings of the National Conference on Artificial
 10749 Intelligence (AAAI), pp. 301–306.
- 10750 Borges, J. L. (1993). Other Inquisitions 1937–1952. University of Texas Press. Translated
 10751 by Ruth L. C. Simms.
- 10752 Botha, J. A. and P. Blunsom (2014). Compositional morphology for word representations
 10753 and language modelling. In Proceedings of the International Conference on Machine
 10754 Learning (ICML).
- 10755 Bottou, L. (2012). Stochastic gradient descent tricks. In Neural networks: Tricks of the
 10756 trade, pp. 421–436. Springer.
- 10757 Bottou, L., F. E. Curtis, and J. Nocedal (2016). Optimization methods for large-scale
 10758 machine learning. arXiv preprint arXiv:1606.04838.
- 10759 Bowman, S. R., L. Vilnis, O. Vinyals, A. Dai, R. Jozefowicz, and S. Bengio (2016).
 10760 Generating sentences from a continuous space. In Proceedings of the Conference on
 10761 Natural Language Learning (CoNLL), pp. 10–21.
- 10762 boyd, d. and K. Crawford (2012). Critical questions for big data. Information,
 10763 Communication & Society 15(5), 662–679.
- 10764 Boyd, S. and L. Vandenberghe (2004). Convex Optimization. New York: Cambridge
 10765 University Press.
- 10766 Branavan, S., H. Chen, J. Eisenstein, and R. Barzilay (2009). Learning document-level
 10767 semantic properties from free-text annotations. Journal of Artificial Intelligence
 10768 Research 34(2), 569–603.
- 10769 Branavan, S. R., H. Chen, L. S. Zettlemoyer, and R. Barzilay (2009). Reinforcement
 10770 learning for mapping instructions to actions. In Proceedings of the Association for
 10771 Computational Linguistics (ACL), pp. 82–90.
- 10772 Braud, C., O. Lacroix, and A. Søgaard (2017). Does syntax help discourse segmentation?
 10773 not so much. In Proceedings of Empirical Methods for Natural Language Processing
 10774 (EMNLP), pp. 2432–2442.
- 10775 Briscoe, T. (2011). Introduction to formal semantics for natural language.

- 10776 Brown, P. F., J. Cocke, S. A. D. Pietra, V. J. D. Pietra, F. Jelinek, J. D. Lafferty,
10777 R. L. Mercer, and P. S. Roossin (1990). A statistical approach to machine translation.
10778 Computational linguistics 16(2), 79–85.
- 10779 Brown, P. F., P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai (1992). Class-based
10780 n-gram models of natural language. Computational linguistics 18(4), 467–479.
- 10781 Brown, P. F., V. J. D. Pietra, S. A. D. Pietra, and R. L. Mercer (1993). The mathematics of
10782 statistical machine translation: Parameter estimation. Computational linguistics 19(2),
10783 263–311.
- 10784 Brun, C. and C. Roux (2014). Décomposition des “hash tags” pour l’amélioration de
10785 la classification en polarité des “tweets”. Proceedings of Traitement Automatique des
10786 Langues Naturelles, 473–478.
- 10787 Bruni, E., N.-K. Tran, and M. Baroni (2014). Multimodal distributional semantics. Journal
10788 of Artificial Intelligence Research 49(2014), 1–47.
- 10789 Bullinaria, J. A. and J. P. Levy (2007). Extracting semantic representations from word
10790 co-occurrence statistics: A computational study. Behavior research methods 39(3), 510–
10791 526.
- 10792 Bunescu, R. C. and R. J. Mooney (2005). A shortest path dependency kernel for relation
10793 extraction. In Proceedings of Empirical Methods for Natural Language Processing
10794 (EMNLP), pp. 724–731.
- 10795 Bunescu, R. C. and M. Pasca (2006). Using encyclopedic knowledge for named entity
10796 disambiguation. In Proceedings of the European Chapter of the Association for
10797 Computational Linguistics (EACL), pp. 9–16.
- 10798 Burstein, J., D. Marcu, and K. Knight (2003). Finding the WRITE stuff: Automatic
10799 identification of discourse structure in student essays. IEEE Intelligent Systems 18(1),
10800 32–39.
- 10801 Burstein, J., J. Tetreault, and S. Andreyev (2010). Using entity-based features to
10802 model coherence in student essays. In Human language technologies: The 2010
10803 annual conference of the North American chapter of the Association for Computational
10804 Linguistics, pp. 681–684. Association for Computational Linguistics.
- 10805 Burstein, J., J. Tetreault, and M. Chodorow (2013). Holistic discourse coherence annotation
10806 for noisy essay writing. Dialogue & Discourse 4(2), 34–52.
- 10807 Cai, Q. and A. Yates (2013). Large-scale semantic parsing via schema matching and lexicon
10808 extension. In Proceedings of the Association for Computational Linguistics (ACL), pp.
10809 423–433.

- 10810 Caliskan, A., J. J. Bryson, and A. Narayanan (2017). Semantics derived automatically
10811 from language corpora contain human-like biases. *Science* 356(6334), 183–186.
- 10812 Canny, J. (1987). A computational approach to edge detection. In *Readings in Computer*
10813 *Vision*, pp. 184–203. Elsevier.
- 10814 Cappé, O. and E. Moulines (2009). On-line expectation–maximization algorithm for
10815 latent data models. *Journal of the Royal Statistical Society: Series B (Statistical*
10816 *Methodology*) 71(3), 593–613.
- 10817 Carbonell, J. and J. Goldstein (1998). The use of mmr, diversity-based reranking
10818 for reordering documents and producing summaries. In *Proceedings of ACM SIGIR*
10819 *conference on Research and development in information retrieval*, pp. 335–336.
- 10820 Carbonell, J. R. (1970). Mixed-initiative man-computer instructional dialogues. Technical
10821 report, BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS.
- 10822 Cardie, C. and K. Wagstaff (1999). Noun phrase coreference as clustering. In *Proceedings*
10823 *of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 82–89.
- 10824 Carletta, J. (1996). Assessing agreement on classification tasks: the kappa statistic.
10825 *Computational linguistics* 22(2), 249–254.
- 10826 Carletta, J. (2007). Unleashing the killer corpus: experiences in creating the
10827 multi-everything ami meeting corpus. *Language Resources and Evaluation* 41(2), 181–
10828 190.
- 10829 Carlson, L. and D. Marcu (2001). Discourse tagging reference manual. Technical Report
10830 ISI-TR-545, Information Sciences Institute.
- 10831 Carlson, L., M. E. Okurowski, and D. Marcu (2002). RST discourse treebank. Linguistic
10832 Data Consortium, University of Pennsylvania.
- 10833 Carpenter, B. (1997). Type-logical semantics. Cambridge, MA: MIT Press.
- 10834 Carreras, X., M. Collins, and T. Koo (2008). Tag, dynamic programming, and the
10835 perceptron for efficient, feature-rich parsing. In *Proceedings of the Conference on Natural*
10836 *Language Learning (CoNLL)*, pp. 9–16.
- 10837 Carreras, X. and L. Màrquez (2005). Introduction to the conll-2005 shared task: Semantic
10838 role labeling. In *Proceedings of the Ninth Conference on Computational Natural*
10839 *Language Learning*, pp. 152–164. Association for Computational Linguistics.
- 10840 Carroll, L. (1865). Alice’s Adventures in Wonderland. London: Macmillan.

- 10841 Carroll, L. (1917). *Through the looking glass: And what Alice found there*. Chicago:
10842 Rand, McNally.
- 10843 Cayley, A. (1889). A theorem on trees. *Quart. J. Math.* 23, 376–378.
- 10844 Chambers, N. and D. Jurafsky (2008). Jointly combining implicit constraints improves
10845 temporal ordering. In *Proceedings of Empirical Methods for Natural Language
10846 Processing (EMNLP)*, pp. 698–706.
- 10847 Chang, K.-W., A. Krishnamurthy, A. Agarwal, H. Daume III, and J. Langford (2015).
10848 Learning to search better than your teacher. In *Proceedings of the International
10849 Conference on Machine Learning (ICML)*.
- 10850 Chang, M.-W., L. Ratinov, and D. Roth (2007). Guiding semi-supervision with
10851 constraint-driven learning. In *Proceedings of the Association for Computational
10852 Linguistics (ACL)*, pp. 280–287.
- 10853 Chang, M.-W., L.-A. Ratinov, N. Rizzolo, and D. Roth (2008). Learning and inference
10854 with constraints. In *Proceedings of the National Conference on Artificial Intelligence
10855 (AAAI)*, pp. 1513–1518.
- 10856 Chapman, W. W., W. Bridewell, P. Hanbury, G. F. Cooper, and B. G. Buchanan (2001).
10857 A simple algorithm for identifying negated findings and diseases in discharge summaries.
10858 *Journal of biomedical informatics* 34(5), 301–310.
- 10859 Charniak, E. (1997). Statistical techniques for natural language parsing. *AI magazine* 18(4),
10860 33–43.
- 10861 Charniak, E. and M. Johnson (2005). Coarse-to-fine n-best parsing and maxent
10862 discriminative reranking. In *Proceedings of the Association for Computational
10863 Linguistics (ACL)*, pp. 173–180.
- 10864 Chelba, C. and A. Acero (2006). Adaptation of maximum entropy capitalizer: Little data
10865 can help a lot. *Computer Speech & Language* 20(4), 382–399.
- 10866 Chelba, C., T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson (2013).
10867 One billion word benchmark for measuring progress in statistical language modeling.
10868 arXiv preprint arXiv:1312.3005.
- 10869 Chen, D., J. Bolton, and C. D. Manning (2016). A thorough examination of the CNN/Daily
10870 Mail reading comprehension task. In *Proceedings of the Association for Computational
10871 Linguistics (ACL)*.
- 10872 Chen, D. and C. D. Manning (2014). A fast and accurate dependency parser using
10873 neural networks. In *Proceedings of Empirical Methods for Natural Language Processing
10874 (EMNLP)*, pp. 740–750.

- 10875 Chen, D. L. and R. J. Mooney (2008). Learning to sportscast: a test of grounded language
 10876 acquisition. In Proceedings of the International Conference on Machine Learning
 10877 (ICML), pp. 128–135.
- 10878 Chen, H., S. Branavan, R. Barzilay, and D. R. Karger (2009). Content modeling using
 10879 latent permutations. *Journal of Artificial Intelligence Research* 36(1), 129–163.
- 10880 Chen, M., Z. Xu, K. Weinberger, and F. Sha (2012). Marginalized denoising autoencoders
 10881 for domain adaptation. In Proceedings of the International Conference on Machine
 10882 Learning (ICML).
- 10883 Chen, M. X., O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones,
 10884 N. Parmar, M. Schuster, Z. Chen, Y. Wu, and M. Hughes (2018). The best of both
 10885 worlds: Combining recent advances in neural machine translation. In Proceedings of the
 10886 Association for Computational Linguistics (ACL).
- 10887 Chen, S. F. and J. Goodman (1999). An empirical study of smoothing techniques for
 10888 language modeling. *Computer Speech & Language* 13(4), 359–393.
- 10889 Chen, T. and C. Guestrin (2016). Xgboost: A scalable tree boosting system. In Proceedings
 10890 of Knowledge Discovery and Data Mining (KDD), pp. 785–794.
- 10891 Chen, X., X. Qiu, C. Zhu, P. Liu, and X. Huang (2015). Long short-term memory
 10892 neural networks for chinese word segmentation. In Proceedings of Empirical Methods
 10893 for Natural Language Processing (EMNLP), pp. 1197–1206.
- 10894 Chen, Y., S. Gilroy, A. Malletti, K. Knight, and J. May (2018). Recurrent neural networks
 10895 as weighted language recognizers. In Proceedings of the North American Chapter of the
 10896 Association for Computational Linguistics (NAACL).
- 10897 Chen, Z. and H. Ji (2009). Graph-based event coreference resolution. In Proceedings of the
 10898 2009 Workshop on Graph-based Methods for Natural Language Processing, pp. 54–57.
 10899 Association for Computational Linguistics.
- 10900 Cheng, X. and D. Roth (2013). Relational inference for wikification. In Proceedings of
 10901 Empirical Methods for Natural Language Processing (EMNLP), pp. 1787–1796.
- 10902 Chiang, D. (2007). Hierarchical phrase-based translation. *Computational Linguistics* 33(2),
 10903 201–228.
- 10904 Chiang, D., J. Graehl, K. Knight, A. Pauls, and S. Ravi (2010). Bayesian inference
 10905 for finite-state transducers. In Proceedings of the North American Chapter of the
 10906 Association for Computational Linguistics (NAACL), pp. 447–455.

- 10907 Chinchor, N. and P. Robinson (1997). Muc-7 named entity task definition. In Proceedings
10908 of the 7th Conference on Message Understanding, Volume 29.
- 10909 Cho, K. (2015). Natural language understanding with distributed representation.
10910 CoRR abs/1511.07916.
- 10911 Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk,
10912 and Y. Bengio (2014). Learning phrase representations using rnn encoder-decoder
10913 for statistical machine translation. In Proceedings of Empirical Methods for Natural
10914 Language Processing (EMNLP).
- 10915 Chomsky, N. (1957). Syntactic structures. The Hague: Mouton & Co.
- 10916 Chomsky, N. (1982). Some concepts and consequences of the theory of government and
10917 binding, Volume 6. MIT press.
- 10918 Choromanska, A., M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun (2015). The loss
10919 surfaces of multilayer networks. In Proceedings of Artificial Intelligence and Statistics
10920 (AISTATS), pp. 192–204.
- 10921 Christensen, J., S. Soderland, O. Etzioni, et al. (2010). Semantic role labeling for open
10922 information extraction. In Proceedings of the Workshop on Formalisms and Methodology
10923 for Learning by Reading, pp. 52–60. Association for Computational Linguistics.
- 10924 Christodoulopoulos, C., S. Goldwater, and M. Steedman (2010). Two decades of
10925 unsupervised pos induction: How far have we come? In Proceedings of Empirical
10926 Methods for Natural Language Processing (EMNLP), pp. 575–584.
- 10927 Chu, Y.-J. and T.-H. Liu (1965). On shortest arborescence of a directed graph. Scientia
10928 Sinica 14(10), 1396–1400.
- 10929 Chung, C. and J. W. Pennebaker (2007). The psychological functions of function words. In
10930 K. Fiedler (Ed.), Social communication, pp. 343–359. New York and Hove: Psychology
10931 Press.
- 10932 Church, K. (2011). A pendulum swung too far. Linguistic Issues in Language
10933 Technology 6(5), 1–27.
- 10934 Church, K. W. (2000). Empirical estimates of adaptation: the chance of two Noriegas is
10935 closer to $p/2$ than p^2 . In Proceedings of the International Conference on Computational
10936 Linguistics (COLING), pp. 180–186.
- 10937 Church, K. W. and P. Hanks (1990). Word association norms, mutual information, and
10938 lexicography. Computational linguistics 16(1), 22–29.

- 10939 Ciaramita, M. and M. Johnson (2003). Supersense tagging of unknown nouns in wordnet.
 10940 In Proceedings of Empirical Methods for Natural Language Processing (EMNLP), pp.
 10941 168–175.
- 10942 Clark, K. and C. D. Manning (2015). Entity-centric coreference resolution with model
 10943 stacking. In Proceedings of the Association for Computational Linguistics (ACL), pp.
 10944 1405–1415.
- 10945 Clark, K. and C. D. Manning (2016). Improving coreference resolution by learning
 10946 entity-level distributed representations. In Proceedings of the Association for
 10947 Computational Linguistics (ACL).
- 10948 Clark, P. (2015). Elementary school science and math tests as a driver for ai: take the
 10949 aristo challenge! In Proceedings of the National Conference on Artificial Intelligence
 10950 (AAAI), pp. 4019–4021.
- 10951 Clarke, J., D. Goldwasser, M.-W. Chang, and D. Roth (2010). Driving semantic parsing
 10952 from the world’s response. In Proceedings of the Conference on Natural Language
 10953 Learning (CoNLL), pp. 18–27.
- 10954 Clarke, J. and M. Lapata (2008). Global inference for sentence compression: An integer
 10955 linear programming approach. *Journal of Artificial Intelligence Research* 31, 399–429.
- 10956 Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and
 10957 psychological measurement* 20(1), 37–46.
- 10958 Cohen, S. (2016). Bayesian analysis in natural language processing. *Synthesis Lectures on
 10959 Human Language Technologies*. San Rafael, CA: Morgan & Claypool Publishers.
- 10960 Collier, N., C. Nobata, and J.-i. Tsujii (2000). Extracting the names of genes and gene
 10961 products with a hidden markov model. In Proceedings of the International Conference
 10962 on Computational Linguistics (COLING), pp. 201–207.
- 10963 Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In
 10964 Proceedings of the Association for Computational Linguistics (ACL), pp. 16–23.
- 10965 Collins, M. (2002). Discriminative training methods for hidden markov models: theory
 10966 and experiments with perceptron algorithms. In Proceedings of Empirical Methods for
 10967 Natural Language Processing (EMNLP), pp. 1–8.
- 10968 Collins, M. (2013). Notes on natural language processing. <http://www.cs.columbia.edu/~mcollins/notes-spring2013.html>.
- 10970 Collins, M. and T. Koo (2005). Discriminative reranking for natural language parsing.
 10971 *Computational Linguistics* 31(1), 25–70.

- 10972 Collins, M. and B. Roark (2004). Incremental parsing with the perceptron algorithm. In
10973 Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics,
10974 pp. 111. Association for Computational Linguistics.
- 10975 Collobert, R., K. Kavukcuoglu, and C. Farabet (2011). Torch7: A matlab-like environment
10976 for machine learning. Technical Report EPFL-CONF-192376, EPFL.
- 10977 Collobert, R. and J. Weston (2008). A unified architecture for natural language processing:
10978 Deep neural networks with multitask learning. In Proceedings of the International
10979 Conference on Machine Learning (ICML), pp. 160–167.
- 10980 Collobert, R., J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa (2011).
10981 Natural language processing (almost) from scratch. *Journal of Machine Learning
10982 Research* 12, 2493–2537.
- 10983 Colton, S., J. Goodwin, and T. Veale (2012). Full-face poetry generation. In Proceedings
10984 of the International Conference on Computational Creativity, pp. 95–102.
- 10985 Conneau, A., D. Kiela, H. Schwenk, L. Barrault, and A. Bordes (2017). Supervised
10986 learning of universal sentence representations from natural language inference data. In
10987 Proceedings of Empirical Methods for Natural Language Processing (EMNLP), pp. 681–
10988 691.
- 10989 Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2009). Introduction to
10990 algorithms (third ed.). MIT press.
- 10991 Cotterell, R., H. Schütze, and J. Eisner (2016). Morphological smoothing and extrapolation
10992 of word embeddings. In Proceedings of the Association for Computational Linguistics
10993 (ACL), pp. 1651–1660.
- 10994 Covello, L., Y. Sohn, A. D. Kramer, C. Marlow, M. Franceschetti, N. A. Christakis, and
10995 J. H. Fowler (2014). Detecting emotional contagion in massive social networks. *PloS
10996 one* 9(3), e90315.
- 10997 Covington, M. A. (2001). A fundamental algorithm for dependency parsing. In Proceedings
10998 of the 39th annual ACM southeast conference, pp. 95–102.
- 10999 Crammer, K., O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer (2006, December).
11000 Online passive-aggressive algorithms. *The Journal of Machine Learning Research* 7,
11001 551–585.
- 11002 Crammer, K. and Y. Singer (2001). Pranking with ranking. In Neural Information
11003 Processing Systems (NIPS), pp. 641–647.

- 11004 Creutz, M. and K. Lagus (2007). Unsupervised models for morpheme segmentation
 11005 and morphology learning. *ACM Transactions on Speech and Language Processing*
 11006 (TSLP) 4(1), 3.
- 11007 Cross, J. and L. Huang (2016). Span-based constituency parsing with a structure-label
 11008 system and provably optimal dynamic oracles. In *Proceedings of Empirical Methods for*
 11009 *Natural Language Processing* (EMNLP), pp. 1–11.
- 11010 Cucerzan, S. (2007). Large-scale named entity disambiguation based on wikipedia data.
 11011 In *Proceedings of Empirical Methods for Natural Language Processing* (EMNLP).
- 11012 Cui, H., R. Sun, K. Li, M.-Y. Kan, and T.-S. Chua (2005). Question answering passage
 11013 retrieval using dependency relations. In *Proceedings of the 28th annual international*
 11014 *ACM SIGIR conference on Research and development in information retrieval*, pp. 400–
 11015 407. ACM.
- 11016 Cui, Y., Z. Chen, S. Wei, S. Wang, T. Liu, and G. Hu (2017). Attention-over-attention
 11017 neural networks for reading comprehension. In *Proceedings of the Association for*
 11018 *Computational Linguistics* (ACL).
- 11019 Culotta, A. and J. Sorensen (2004). Dependency tree kernels for relation extraction. In
 11020 *Proceedings of the Association for Computational Linguistics* (ACL).
- 11021 Culotta, A., M. Wick, and A. McCallum (2007). First-order probabilistic models for
 11022 coreference resolution. In *Proceedings of the North American Chapter of the Association*
 11023 *for Computational Linguistics* (NAACL), pp. 81–88.
- 11024 Curry, H. B. and R. Feys (1958). *Combinatory Logic, Volume I*. Amsterdam: North
 11025 Holland.
- 11026 Danescu-Niculescu-Mizil, C., M. Sudhof, D. Jurafsky, J. Leskovec, and C. Potts (2013). A
 11027 computational approach to politeness with application to social factors. In *Proceedings*
 11028 *of the Association for Computational Linguistics* (ACL), pp. 250–259.
- 11029 Das, D., D. Chen, A. F. Martins, N. Schneider, and N. A. Smith (2014). Frame-semantic
 11030 parsing. *Computational Linguistics* 40(1), 9–56.
- 11031 Daumé III, H. (2007). Frustratingly easy domain adaptation. In *Proceedings of the*
 11032 *Association for Computational Linguistics* (ACL).
- 11033 Daumé III, H., J. Langford, and D. Marcu (2009). Search-based structured prediction.
 11034 *Machine learning* 75(3), 297–325.
- 11035 Daumé III, H. and D. Marcu (2005). A large-scale exploration of effective global features
 11036 for a joint entity detection and tracking model. In *Proceedings of Empirical Methods*
 11037 *for Natural Language Processing* (EMNLP), pp. 97–104.

- 11038 Dauphin, Y. N., R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio (2014).
11039 Identifying and attacking the saddle point problem in high-dimensional non-convex
11040 optimization. In Neural Information Processing Systems (NIPS), pp. 2933–2941.
- 11041 Davidson, D. (1967). The logical form of action sentences. In N. Rescher (Ed.), *The Logic*
11042 of Decision and Action. Pittsburgh: University of Pittsburgh Press.
- 11043 De Gispert, A. and J. B. Marino (2006). Catalan-english statistical machine translation
11044 without parallel corpus: bridging through spanish. In Proc. of 5th International
11045 Conference on Language Resources and Evaluation (LREC), pp. 65–68. Citeseer.
- 11046 De Marneffe, M.-C. and C. D. Manning (2008). The stanford typed dependencies
11047 representation. In Coling 2008: Proceedings of the workshop on Cross-Framework and
11048 Cross-Domain Parser Evaluation, pp. 1–8. Association for Computational Linguistics.
- 11049 Dean, J. and S. Ghemawat (2008). Mapreduce: simplified data processing on large clusters.
11050 Communications of the ACM 51(1), 107–113.
- 11051 Deerwester, S. C., S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman
11052 (1990). Indexing by latent semantic analysis. JASIS 41(6), 391–407.
- 11053 Dehdari, J. (2014). A Neurophysiologically-Inspired Statistical Language Model. Ph. D.
11054 thesis, The Ohio State University.
- 11055 Deisenroth, M. P., A. A. Faisal, and C. S. Ong (2018). Mathematics For Machine Learning.
11056 Cambridge UP.
- 11057 Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). Maximum likelihood from
11058 incomplete data via the em algorithm. Journal of the Royal Statistical Society. Series B
11059 (Methodological), 1–38.
- 11060 Denis, P. and J. Baldridge (2007). A ranking approach to pronoun resolution. In IJCAI.
- 11061 Denis, P. and J. Baldridge (2008). Specialized models and ranking for coreference
11062 resolution. In Proceedings of the Conference on Empirical Methods in Natural
11063 Language Processing, EMNLP '08, Stroudsburg, PA, USA, pp. 660–669. Association
11064 for Computational Linguistics.
- 11065 Denis, P. and J. Baldridge (2009). Global joint models for coreference resolution and named
11066 entity classification. Procesamiento del Lenguaje Natural 42.
- 11067 Derrida, J. (1985). Des tours de babel. In J. Graham (Ed.), *Difference in translation*.
11068 Ithaca, NY: Cornell University Press.

- 11069 Dhingra, B., H. Liu, Z. Yang, W. W. Cohen, and R. Salakhutdinov (2017). Gated-attention
11070 readers for text comprehension. In Proceedings of the Association for Computational
11071 Linguistics (ACL).
- 11072 Diaconis, P. and B. Skyrms (2017). Ten Great Ideas About Chance. Princeton University
11073 Press.
- 11074 Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised
11075 classification learning algorithms. *Neural computation* 10(7), 1895–1923.
- 11076 Dietterich, T. G., R. H. Lathrop, and T. Lozano-Pérez (1997). Solving the multiple instance
11077 problem with axis-parallel rectangles. *Artificial intelligence* 89(1), 31–71.
- 11078 Dimitrova, L., N. Ide, V. Petkevici, T. Erjavec, H. J. Kaalep, and D. Tufis (1998).
11079 Multext-east: Parallel and comparable corpora and lexicons for six central and
11080 eastern european languages. In Proceedings of the 17th international conference
11081 on Computational linguistics-Volume 1, pp. 315–319. Association for Computational
11082 Linguistics.
- 11083 Doddington, G. R., A. Mitchell, M. A. Przybocki, L. A. Ramshaw, S. Strassel, and R. M.
11084 Weischedel (2004). The automatic content extraction (ace) program-tasks, data, and
11085 evaluation. In Proceedings of the Language Resources and Evaluation Conference, pp.
11086 837–840.
- 11087 dos Santos, C., B. Xiang, and B. Zhou (2015). Classifying relations by ranking with
11088 convolutional neural networks. In Proceedings of the Association for Computational
11089 Linguistics (ACL), pp. 626–634.
- 11090 Dowty, D. (1991). Thematic proto-roles and argument selection. *Language*, 547–619.
- 11091 Dredze, M., P. McNamee, D. Rao, A. Gerber, and T. Finin (2010). Entity disambiguation
11092 for knowledge base population. In Proceedings of the 23rd International Conference on
11093 Computational Linguistics, pp. 277–285. Association for Computational Linguistics.
- 11094 Dredze, M., M. J. Paul, S. Bergsma, and H. Tran (2013). Carmen: A Twitter geolocation
11095 system with applications to public health. In AAAI workshop on expanding the
11096 boundaries of health informatics using AI (HIAI), pp. 20–24.
- 11097 Dreyfus, H. L. (1992). What computers still can't do: a critique of artificial reason. MIT
11098 press.
- 11099 Du, L., W. Buntine, and M. Johnson (2013). Topic segmentation with a structured
11100 topic model. In Proceedings of the North American Chapter of the Association for
11101 Computational Linguistics (NAACL), pp. 190–200.

- 11102 Duchi, J., E. Hazan, and Y. Singer (2011). Adaptive subgradient methods for online
11103 learning and stochastic optimization. *The Journal of Machine Learning Research* 12,
11104 2121–2159.
- 11105 Dunietz, J., L. Levin, and J. Carbonell (2017). The because corpus 2.0: Annotating
11106 causality and overlapping relations. In *Proceedings of the Linguistic Annotation*
11107 *Workshop*.
- 11108 Durrett, G., T. Berg-Kirkpatrick, and D. Klein (2016). Learning-based single-document
11109 summarization with compression and anaphoricity constraints. In *Proceedings of the*
11110 *Association for Computational Linguistics (ACL)*, pp. 1998–2008.
- 11111 Durrett, G. and D. Klein (2013). Easy victories and uphill battles in coreference resolution.
11112 In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.
- 11113 Durrett, G. and D. Klein (2015). Neural crf parsing. In *Proceedings of the Association for*
11114 *Computational Linguistics (ACL)*.
- 11115 Dyer, C., M. Ballesteros, W. Ling, A. Matthews, and N. A. Smith (2015). Transition-based
11116 dependency parsing with stack long short-term memory. In *Proceedings of the*
11117 *Association for Computational Linguistics (ACL)*, pp. 334–343.
- 11118 Dyer, C., A. Kuncoro, M. Ballesteros, and N. A. Smith (2016). Recurrent neural network
11119 grammars. In *Proceedings of the North American Chapter of the Association for*
11120 *Computational Linguistics (NAACL)*, pp. 199–209.
- 11121 Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of*
11122 *Standards B* 71(4), 233–240.
- 11123 Efron, B. and R. J. Tibshirani (1993). An introduction to the bootstrap: Monographs
11124 on statistics and applied probability, vol. 57. New York and London: Chapman and
11125 Hall/CRC.
- 11126 Eisenstein, J. (2009). Hierarchical text segmentation from multi-scale lexical cohesion.
11127 In *Proceedings of the North American Chapter of the Association for Computational*
11128 *Linguistics (NAACL)*.
- 11129 Eisenstein, J. and R. Barzilay (2008). Bayesian unsupervised topic segmentation. In
11130 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11131 Eisner, J. (1997). State-of-the-art algorithms for minimum spanning trees: A tutorial
11132 discussion.
- 11133 Eisner, J. (2000). Bilexical grammars and their cubic-time parsing algorithms. In *Advances*
11134 *in probabilistic and other parsing technologies*, pp. 29–61. Springer.

- 11135 Eisner, J. (2002). Parameter estimation for probabilistic finite-state transducers. In
 11136 Proceedings of the Association for Computational Linguistics (ACL), pp. 1–8.
- 11137 Eisner, J. (2016). Inside-outside and forward-backward algorithms are just backprop. In
 11138 Proceedings of the Workshop on Structured Prediction for NLP, pp. 1–17.
- 11139 Eisner, J. M. (1996). Three new probabilistic models for dependency parsing: An
 11140 exploration. In Proceedings of the International Conference on Computational
 11141 Linguistics (COLING), pp. 340–345.
- 11142 Ekman, P. (1992). Are there basic emotions? *Psychological Review* 99(3), 550–553.
- 11143 Elman, J. L. (1990). Finding structure in time. *Cognitive science* 14(2), 179–211.
- 11144 Elman, J. L., E. A. Bates, M. H. Johnson, A. Karmiloff-Smith, D. Parisi, and K. Plunkett
 11145 (1998). Rethinking innateness: A connectionist perspective on development, Volume 10.
 11146 MIT press.
- 11147 Elsner, M. and E. Charniak (2010). Disentangling chat. *Computational Linguistics* 36(3),
 11148 389–409.
- 11149 Esuli, A. and F. Sebastiani (2006). Sentiwordnet: A publicly available lexical resource for
 11150 opinion mining. In LREC, Volume 6, pp. 417–422. Citeseer.
- 11151 Etzioni, O., A. Fader, J. Christensen, S. Soderland, and M. Mausam (2011). Open
 11152 information extraction: The second generation. In Proceedings of the International
 11153 Joint Conference on Artificial Intelligence (IJCAI), pp. 3–10.
- 11154 Faruqui, M., J. Dodge, S. K. Jauhar, C. Dyer, E. Hovy, and N. A. Smith (2015). Retrofitting
 11155 word vectors to semantic lexicons. In Proceedings of the North American Chapter of the
 11156 Association for Computational Linguistics (NAACL).
- 11157 Faruqui, M. and C. Dyer (2014). Improving vector space word representations using
 11158 multilingual correlation. In Proceedings of the European Chapter of the Association
 11159 for Computational Linguistics (EACL), pp. 462–471.
- 11160 Faruqui, M., R. McDonald, and R. Soricut (2016). Morpho-syntactic lexicon generation
 11161 using graph-based semi-supervised learning. *Transactions of the Association for
 11162 Computational Linguistics* 4, 1–16.
- 11163 Faruqui, M., Y. Tsvetkov, P. Rastogi, and C. Dyer (2016, August). Problems with
 11164 evaluation of word embeddings using word similarity tasks. In Proceedings of the 1st
 11165 Workshop on Evaluating Vector-Space Representations for NLP, Berlin, Germany, pp.
 11166 30–35. Association for Computational Linguistics.
- 11167 Fellbaum, C. (2010). WordNet. Springer.

- 11168 Feng, V. W., Z. Lin, and G. Hirst (2014). The impact of deep hierarchical discourse
11169 structures in the evaluation of text coherence. In Proceedings of the International
11170 Conference on Computational Linguistics (COLING), pp. 940–949.
- 11171 Feng, X., L. Huang, D. Tang, H. Ji, B. Qin, and T. Liu (2016). A language-independent
11172 neural network for event detection. In Proceedings of the Association for Computational
11173 Linguistics (ACL), pp. 66–71.
- 11174 Fernandes, E. R., C. N. dos Santos, and R. L. Milidiú (2014). Latent trees for coreference
11175 resolution. Computational Linguistics.
- 11176 Ferrucci, D., E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally,
11177 J. W. Murdock, E. Nyberg, J. Prager, et al. (2010). Building Watson: An overview of
11178 the DeepQA project. *AI magazine* 31(3), 59–79.
- 11179 Ficler, J. and Y. Goldberg (2017, September). Controlling linguistic style aspects in neural
11180 language generation. In Proceedings of the Workshop on Stylistic Variation, Copenhagen,
11181 Denmark, pp. 94–104. Association for Computational Linguistics.
- 11182 Filippova, K. and M. Strube (2008). Sentence fusion via dependency graph compression.
11183 In Proceedings of Empirical Methods for Natural Language Processing (EMNLP), pp.
11184 177–185.
- 11185 Fillmore, C. J. (1968). The case for case. In E. Bach and R. Harms (Eds.), *Universals in*
11186 *linguistic theory*. Holt, Rinehart, and Winston.
- 11187 Fillmore, C. J. (1976). Frame semantics and the nature of language. *Annals of the New*
11188 *York Academy of Sciences* 280(1), 20–32.
- 11189 Fillmore, C. J. and C. Baker (2009). A frames approach to semantic analysis. In *The*
11190 *Oxford Handbook of Linguistic Analysis*. Oxford University Press.
- 11191 Finkel, J. R., T. Grenager, and C. Manning (2005). Incorporating non-local information
11192 into information extraction systems by gibbs sampling. In Proceedings of the Association
11193 for Computational Linguistics (ACL), pp. 363–370.
- 11194 Finkel, J. R., T. Grenager, and C. D. Manning (2007). The infinite tree. In Proceedings
11195 of the Association for Computational Linguistics (ACL), pp. 272–279.
- 11196 Finkel, J. R., A. Kleeman, and C. D. Manning (2008). Efficient, feature-based, conditional
11197 random field parsing. In Proceedings of the Association for Computational Linguistics
11198 (ACL), pp. 959–967.
- 11199 Finkel, J. R. and C. Manning (2009). Hierarchical bayesian domain adaptation. In
11200 Proceedings of the North American Chapter of the Association for Computational
11201 Linguistics (NAACL), pp. 602–610.

- 11202 Finkel, J. R. and C. D. Manning (2008). Enforcing transitivity in coreference resolution.
 11203 In Proceedings of the 46th Annual Meeting of the Association for Computational
 11204 Linguistics on Human Language Technologies: Short Papers, pp. 45–48. Association
 11205 for Computational Linguistics.
- 11206 Finkelstein, L., E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin
 11207 (2002). Placing search in context: The concept revisited. ACM Transactions on
 11208 Information Systems 20(1), 116–131.
- 11209 Firth, J. R. (1957). Papers in Linguistics 1934-1951. Oxford University Press.
- 11210 Flanigan, J., S. Thomson, J. Carbonell, C. Dyer, and N. A. Smith (2014). A discriminative
 11211 graph-based parser for the abstract meaning representation. In Proceedings of the
 11212 Association for Computational Linguistics (ACL), pp. 1426–1436.
- 11213 Foltz, P. W., W. Kintsch, and T. K. Landauer (1998). The measurement of textual
 11214 coherence with latent semantic analysis. Discourse processes 25(2-3), 285–307.
- 11215 Fordyce, C. S. (2007). Overview of the iwslt 2007 evaluation campaign. In International
 11216 Workshop on Spoken Language Translation (IWSLT) 2007.
- 11217 Fox, H. (2002). Phrasal cohesion and statistical machine translation. In Proceedings of
 11218 Empirical Methods for Natural Language Processing (EMNLP), pp. 304–3111.
- 11219 Francis, W. and H. Kucera (1982). Frequency analysis of English usage. Houghton Mifflin
 11220 Company.
- 11221 Francis, W. N. (1964). A standard sample of present-day English for use with digital
 11222 computers. Report to the U.S Office of Education on Cooperative Research Project No.
 11223 E-007.
- 11224 Freund, Y. and R. E. Schapire (1999). Large margin classification using the perceptron
 11225 algorithm. Machine learning 37(3), 277–296.
- 11226 Fromkin, V., R. Rodman, and N. Hyams (2013). An introduction to language. Cengage
 11227 Learning.
- 11228 Fundel, K., R. Küffner, and R. Zimmer (2007). Relex – relation extraction using
 11229 dependency parse trees. Bioinformatics 23(3), 365–371.
- 11230 Gabow, H. N., Z. Galil, T. Spencer, and R. E. Tarjan (1986). Efficient algorithms for
 11231 finding minimum spanning trees in undirected and directed graphs. Combinatorica 6(2),
 11232 109–122.

- 11233 Gabrilovich, E. and S. Markovitch (2007). Computing semantic relatedness using
11234 wikipedia-based explicit semantic analysis. In Proceedings of the International Joint
11235 Conference on Artificial Intelligence (IJCAI), Volume 7, pp. 1606–1611.
- 11236 Gage, P. (1994). A new algorithm for data compression. *The C Users Journal* 12(2), 23–38.
- 11237 Gale, W. A., K. W. Church, and D. Yarowsky (1992). One sense per discourse. In
11238 Proceedings of the workshop on Speech and Natural Language, pp. 233–237. Association
11239 for Computational Linguistics.
- 11240 Galley, M., M. Hopkins, K. Knight, and D. Marcu (2004). What's in a translation rule?
11241 In Proceedings of the North American Chapter of the Association for Computational
11242 Linguistics (NAACL), pp. 273–280.
- 11243 Galley, M., K. R. McKeown, E. Fosler-Lussier, and H. Jing (2003). Discourse segmentation
11244 of multi-party conversation. In Proceedings of the Association for Computational
11245 Linguistics (ACL).
- 11246 Ganchev, K. and M. Dredze (2008). Small statistical models by random feature mixing. In
11247 Proceedings of the ACL08 HLT Workshop on Mobile Language Processing, pp. 19–20.
- 11248 Ganchev, K., J. Graça, J. Gillenwater, and B. Taskar (2010). Posterior regularization
11249 for structured latent variable models. *The Journal of Machine Learning Research* 11,
11250 2001–2049.
- 11251 Ganin, Y., E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand,
11252 and V. Lempitsky (2016). Domain-adversarial training of neural networks. *Journal of
11253 Machine Learning Research* 17(59), 1–35.
- 11254 Gao, J., G. Andrew, M. Johnson, and K. Toutanova (2007). A comparative study of
11255 parameter estimation methods for statistical natural language processing. In Proceedings
11256 of the Association for Computational Linguistics (ACL), pp. 824–831.
- 11257 Gatt, A. and E. Krahmer (2018). Survey of the state of the art in natural language
11258 generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence
11259 Research* 61, 65–170.
- 11260 Gatt, A. and E. Reiter (2009). Simplenlg: A realisation engine for practical applications.
11261 In Proceedings of the 12th European Workshop on Natural Language Generation, pp.
11262 90–93. Association for Computational Linguistics.
- 11263 Ge, D., X. Jiang, and Y. Ye (2011). A note on the complexity of l_p minimization.
11264 Mathematical programming 129(2), 285–299.

- 11265 Ge, N., J. Hale, and E. Charniak (1998). A statistical approach to anaphora resolution. In
 11266 Proceedings of the sixth workshop on very large corpora, Volume 71, pp. 76.
- 11267 Ge, R., F. Huang, C. Jin, and Y. Yuan (2015). Escaping from saddle points — online
 11268 stochastic gradient for tensor decomposition. In P. Grünwald, E. Hazan, and S. Kale
 11269 (Eds.), *Proceedings of the Conference On Learning Theory (COLT)*.
- 11270 Ge, R. and R. J. Mooney (2005). A statistical semantic parser that integrates syntax and
 11271 semantics. In *Proceedings of the Conference on Natural Language Learning (CoNLL)*,
 11272 pp. 9–16.
- 11273 Geach, P. T. (1962). Reference and generality: An examination of some medieval and
 11274 modern theories. Cornell University Press.
- 11275 Gildea, D. and D. Jurafsky (2002). Automatic labeling of semantic roles. *Computational
 11276 linguistics* 28(3), 245–288.
- 11277 Gimpel, K., N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman,
 11278 D. Yogatama, J. Flanigan, and N. A. Smith (2011). Part-of-speech tagging for
 11279 Twitter: annotation, features, and experiments. In *Proceedings of the Association for
 11280 Computational Linguistics (ACL)*, pp. 42–47.
- 11281 Glass, J., T. J. Hazen, S. Cyphers, I. Malioutov, D. Huynh, and R. Barzilay (2007). Recent
 11282 progress in the mit spoken lecture processing project. In *Eighth Annual Conference of
 11283 the International Speech Communication Association*.
- 11284 Glorot, X. and Y. Bengio (2010). Understanding the difficulty of training deep feedforward
 11285 neural networks. In *Proceedings of Artificial Intelligence and Statistics (AISTATS)*, pp.
 11286 249–256.
- 11287 Glorot, X., A. Bordes, and Y. Bengio (2011). Deep sparse rectifier networks. In *Proceedings
 11288 of the 14th International Conference on Artificial Intelligence and Statistics. JMLR
 11289 W&CP Volume*, Volume 15, pp. 315–323.
- 11290 Godfrey, J. J., E. C. Holliman, and J. McDaniel (1992). Switchboard: Telephone speech
 11291 corpus for research and development. In *Proceedings of the International Conference on
 11292 Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 517–520. IEEE.
- 11293 Goldberg, Y. (2017a, June). An adversarial review of “adversarial generation of natural
 11294 language”. <https://medium.com/@yoav.goldberg/an-adversarial-review-of-adversarial-generation-of-natural-language-4f3a2a2a2a2a>
- 11295 Goldberg, Y. (2017b). Neural Network Methods for Natural Language Processing.
 11296 *Synthesis Lectures on Human Language Technologies*. Morgan & Claypool Publishers.

- 11297 Goldberg, Y. and M. Elhadad (2010). An efficient algorithm for easy-first non-directional
11298 dependency parsing. In Proceedings of the North American Chapter of the Association
11299 for Computational Linguistics (NAACL), pp. 742–750.
- 11300 Goldberg, Y. and J. Nivre (2012). A dynamic oracle for arc-eager dependency parsing. In
11301 Proceedings of the International Conference on Computational Linguistics (COLING),
11302 pp. 959–976.
- 11303 Goldberg, Y., K. Zhao, and L. Huang (2013). Efficient implementation of beam-search
11304 incremental parsers. In ACL (2), pp. 628–633.
- 11305 Goldwater, S. and T. Griffiths (2007). A fully bayesian approach to unsupervised
11306 part-of-speech tagging. In Annual meeting-association for computational linguistics,
11307 Volume 45.
- 11308 Gonçalo Oliveira, H. R., F. A. Cardoso, and F. C. Pereira (2007). Tra-la-lyrics: An
11309 approach to generate text based on rhythm. In Proceedings of the 4th. International
11310 Joint Workshop on Computational Creativity. A. Cardoso and G. Wiggins.
- 11311 Goodfellow, I., Y. Bengio, and A. Courville (2016). Deep learning. MIT Press.
- 11312 Goodman, J. T. (2001). A bit of progress in language modeling. Computer Speech &
11313 Language 15(4), 403–434.
- 11314 Gouws, S., D. Metzler, C. Cai, and E. Hovy (2011). Contextual bearing on linguistic
11315 variation in social media. In LASM.
- 11316 Goyal, A., H. Daume III, and S. Venkatasubramanian (2009). Streaming for large scale nlp:
11317 Language modeling. In Proceedings of the North American Chapter of the Association
11318 for Computational Linguistics (NAACL), pp. 512–520.
- 11319 Graves, A. (2012). Sequence transduction with recurrent neural networks. In Proceedings
11320 of the International Conference on Machine Learning (ICML).
- 11321 Graves, A. and N. Jaitly (2014). Towards end-to-end speech recognition with recurrent
11322 neural networks. In Proceedings of the International Conference on Machine Learning
11323 (ICML), pp. 1764–1772.
- 11324 Graves, A. and J. Schmidhuber (2005). Framewise phoneme classification with bidirectional
11325 lstm and other neural network architectures. Neural Networks 18(5), 602–610.
- 11326 Grice, H. P. (1975). Logic and conversation. In P. Cole and J. L. Morgan (Eds.), Syntax
11327 and Semantics Volume 3: Speech Acts, pp. 41–58. Academic Press.

- 11328 Grishman, R. (2012). Information extraction: Capabilities and challenges. Notes prepared
 11329 for the 2012 International Winter School in Language and Speech Technologies, Rovira
 11330 i Virgili University, Tarragona, Spain.
- 11331 Grishman, R. (2015). Information extraction. *IEEE Intelligent Systems* 30(5), 8–15.
- 11332 Grishman, R., C. Macleod, and J. Sterling (1992). Evaluating parsing strategies using
 11333 standardized parse files. In *Proceedings of the third conference on Applied natural
 11334 language processing*, pp. 156–161. Association for Computational Linguistics.
- 11335 Grishman, R. and B. Sundheim (1996). Message understanding conference-6: A brief
 11336 history. In *Proceedings of the International Conference on Computational Linguistics*
 11337 (COLING), pp. 466–471.
- 11338 Groenendijk, J. and M. Stokhof (1991). Dynamic predicate logic. *Linguistics and
 11339 philosophy* 14(1), 39–100.
- 11340 Grosz, B. J. (1979). Focusing and description in natural language dialogues. Technical
 11341 report, SRI INTERNATIONAL MENLO PARK CA.
- 11342 Grosz, B. J., S. Weinstein, and A. K. Joshi (1995). Centering: A framework for modeling
 11343 the local coherence of discourse. *Computational linguistics* 21(2), 203–225.
- 11344 Gu, J., Z. Lu, H. Li, and V. O. Li (2016). Incorporating copying mechanism in
 11345 sequence-to-sequence learning. In *Proceedings of the Association for Computational
 11346 Linguistics (ACL)*, pp. 1631–1640.
- 11347 Gulcehre, C., S. Ahn, R. Nallapati, B. Zhou, and Y. Bengio (2016). Pointing the unknown
 11348 words. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
 11349 140–149.
- 11350 Gutmann, M. U. and A. Hyvärinen (2012). Noise-contrastive estimation of unnormalized
 11351 statistical models, with applications to natural image statistics. *The Journal of Machine
 11352 Learning Research* 13(1), 307–361.
- 11353 Haghghi, A. and D. Klein (2007). Unsupervised coreference resolution in a nonparametric
 11354 bayesian model. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11355 Haghghi, A. and D. Klein (2009). Simple coreference resolution with rich syntactic and
 11356 semantic features. In *Proceedings of Empirical Methods for Natural Language Processing
 11357 (EMNLP)*, pp. 1152–1161.
- 11358 Haghghi, A. and D. Klein (2010). Coreference resolution in a modular, entity-centered
 11359 model. In *Proceedings of the North American Chapter of the Association for
 11360 Computational Linguistics (NAACL)*, pp. 385–393.

- 11361 Hajič, J. and B. Hladká (1998). Tagging inflective languages: Prediction of morphological
11362 categories for a rich, structured tagset. In Proceedings of the Association for
11363 Computational Linguistics (ACL), pp. 483–490.
- 11364 Halliday, M. and R. Hasan (1976). Cohesion in English. London: Longman.
- 11365 Hammerton, J. (2003). Named entity recognition with long short-term memory. In
11366 Proceedings of the Conference on Natural Language Learning (CoNLL), pp. 172–175.
- 11367 Han, X. and L. Sun (2012). An entity-topic model for entity linking. In Proceedings of
11368 Empirical Methods for Natural Language Processing (EMNLP), pp. 105–115.
- 11369 Han, X., L. Sun, and J. Zhao (2011). Collective entity linking in web text: a graph-based
11370 method. In Proceedings of ACM SIGIR conference on Research and development in
11371 information retrieval, pp. 765–774.
- 11372 Hannak, A., E. Anderson, L. F. Barrett, S. Lehmann, A. Mislove, and M. Riedewald (2012).
11373 Tweetin’ in the rain: Exploring societal-scale effects of weather on mood. In Proceedings
11374 of the International Conference on Web and Social Media (ICWSM).
- 11375 Hardmeier, C. (2012). Discourse in statistical machine translation. a survey and a case
11376 study. Discours. Revue de linguistique, psycholinguistique et informatique. A journal of
11377 linguistics, psycholinguistics and computational linguistics (11).
- 11378 Haspelmath, M. and A. Sims (2013). Understanding morphology. Routledge.
- 11379 Hastie, T., R. Tibshirani, and J. Friedman (2009). The elements of statistical learning
11380 (Second ed.). New York: Springer.
- 11381 Hatzivassiloglou, V. and K. R. McKeown (1997). Predicting the semantic orientation of
11382 adjectives. In Proceedings of the Association for Computational Linguistics (ACL), pp.
11383 174–181.
- 11384 Hayes, A. F. and K. Krippendorff (2007). Answering the call for a standard reliability
11385 measure for coding data. Communication methods and measures 1(1), 77–89.
- 11386 He, H., A. Balakrishnan, M. Eric, and P. Liang (2017). Learning symmetric collaborative
11387 dialogue agents with dynamic knowledge graph embeddings. In Proceedings of the
11388 Association for Computational Linguistics (ACL), pp. 1766–1776.
- 11389 He, K., X. Zhang, S. Ren, and J. Sun (2015). Delving deep into rectifiers: Surpassing
11390 human-level performance on imagenet classification. In Proceedings of the International
11391 Conference on Computer Vision (ICCV), pp. 1026–1034.

- 11392 He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep residual learning for image recognition.
 11393 In Proceedings of the International Conference on Computer Vision (ICCV), pp. 770–
 11394 778.
- 11395 He, L., K. Lee, M. Lewis, and L. Zettlemoyer (2017). Deep semantic role labeling: What
 11396 works and what’s next. In Proceedings of the Association for Computational Linguistics
 11397 (ACL).
- 11398 He, Z., S. Liu, M. Li, M. Zhou, L. Zhang, and H. Wang (2013). Learning entity
 11399 representation for entity disambiguation. In Proceedings of the Association for
 11400 Computational Linguistics (ACL), pp. 30–34.
- 11401 Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. In
 11402 Proceedings of the International Conference on Computational Linguistics (COLING),
 11403 pp. 539–545. Association for Computational Linguistics.
- 11404 Hearst, M. A. (1997). Texttiling: Segmenting text into multi-paragraph subtopic passages.
 11405 Computational linguistics 23(1), 33–64.
- 11406 Heerschop, B., F. Goossen, A. Hogenboom, F. Frasincar, U. Kaymak, and F. de Jong
 11407 (2011). Polarity analysis of texts using discourse structure. In Proceedings of the 20th
 11408 ACM international conference on Information and knowledge management, pp. 1061–
 11409 1070. ACM.
- 11410 Henderson, J. (2004). Discriminative training of a neural network statistical parser. In
 11411 Proceedings of the Association for Computational Linguistics (ACL), pp. 95–102.
- 11412 Hendrickx, I., S. N. Kim, Z. Kozareva, P. Nakov, D. Ó Séaghdha, S. Padó, M. Pennacchiotti,
 11413 L. Romano, and S. Szpakowicz (2009). SemEval-2010 task 8: Multi-way classification
 11414 of semantic relations between pairs of nominals. In Proceedings of the Workshop
 11415 on Semantic Evaluations: Recent Achievements and Future Directions, pp. 94–99.
 11416 Association for Computational Linguistics.
- 11417 Hermann, K. M., T. Kočiský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and
 11418 P. Blunsom (2015). Teaching machines to read and comprehend. In Advances in Neural
 11419 Information Processing Systems, pp. 1693–1701.
- 11420 Hernault, H., H. Prendinger, D. A. duVerle, and M. Ishizuka (2010). HILDA: A discourse
 11421 parser using support vector machine classification. Dialogue and Discourse 1(3), 1–33.
- 11422 Hill, F., A. Bordes, S. Chopra, and J. Weston (2016). The goldilocks principle:
 11423 Reading children’s books with explicit memory representations. In Proceedings of the
 11424 International Conference on Learning Representations (ICLR).

- 11425 Hill, F., K. Cho, and A. Korhonen (2016). Learning distributed representations of sentences
11426 from unlabelled data. In Proceedings of the North American Chapter of the Association
11427 for Computational Linguistics (NAACL).
- 11428 Hindle, D. and M. Rooth (1993). Structural ambiguity and lexical relations. Computational
11429 linguistics 19(1), 103–120.
- 11430 Hirao, T., Y. Yoshida, M. Nishino, N. Yasuda, and M. Nagata (2013). Single-document
11431 summarization as a tree knapsack problem. In Proceedings of Empirical Methods for
11432 Natural Language Processing (EMNLP), pp. 1515–1520.
- 11433 Hirschman, L. and R. Gaizauskas (2001). Natural language question answering: the view
11434 from here. natural language engineering 7(4), 275–300.
- 11435 Hirschman, L., M. Light, E. Breck, and J. D. Burger (1999). Deep read: A reading
11436 comprehension system. In Proceedings of the Association for Computational Linguistics
11437 (ACL), pp. 325–332.
- 11438 Hobbs, J. R. (1978). Resolving pronoun references. Lingua 44(4), 311–338.
- 11439 Hobbs, J. R., D. Appelt, J. Bear, D. Israel, M. Kameyama, M. Stickel, and M. Tyson
11440 (1997). Fastus: A cascaded finite-state transducer for extracting information from
11441 natural-language text. Finite-state language processing, 383–406.
- 11442 Hochreiter, S. and J. Schmidhuber (1997). Long short-term memory. Neural
11443 computation 9(8), 1735–1780.
- 11444 Hockenmaier, J. and M. Steedman (2007). Ccgbank: a corpus of ccg derivations
11445 and dependency structures extracted from the penn treebank. Computational
11446 Linguistics 33(3), 355–396.
- 11447 Hoffart, J., M. A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, B. Taneva,
11448 S. Thater, and G. Weikum (2011). Robust disambiguation of named entities in text.
11449 In Proceedings of Empirical Methods for Natural Language Processing (EMNLP), pp.
11450 782–792.
- 11451 Hoffmann, R., C. Zhang, X. Ling, L. Zettlemoyer, and D. S. Weld (2011). Knowledge-based
11452 weak supervision for information extraction of overlapping relations. In Proceedings of
11453 the Association for Computational Linguistics (ACL), pp. 541–550.
- 11454 Holmstrom, L. and P. Koistinen (1992). Using additive noise in back-propagation training.
11455 IEEE Transactions on Neural Networks 3(1), 24–38.
- 11456 Hovy, E. and J. Lavid (2010). Towards a 'science' of corpus annotation: a new
11457 methodological challenge for corpus linguistics. International journal of translation 22(1),
11458 13–36.

- 11459 Hsu, D., S. M. Kakade, and T. Zhang (2012). A spectral algorithm for learning hidden
 11460 markov models. *Journal of Computer and System Sciences* 78(5), 1460–1480.
- 11461 Hu, M. and B. Liu (2004). Mining and summarizing customer reviews. In Proceedings of
 11462 Knowledge Discovery and Data Mining (KDD), pp. 168–177.
- 11463 Hu, Z., Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing (2017). Toward controlled
 11464 generation of text. In International Conference on Machine Learning, pp. 1587–1596.
- 11465 Huang, F. and A. Yates (2012). Biased representation learning for domain adaptation.
 11466 In Proceedings of Empirical Methods for Natural Language Processing (EMNLP), pp.
 11467 1313–1323.
- 11468 Huang, L. and D. Chiang (2007). Forest rescoring: Faster decoding with integrated
 11469 language models. In Proceedings of the Association for Computational Linguistics
 11470 (ACL), pp. 144–151.
- 11471 Huang, L., S. Fayong, and Y. Guo (2012). Structured perceptron with inexact search.
 11472 In Proceedings of the North American Chapter of the Association for Computational
 11473 Linguistics (NAACL), pp. 142–151.
- 11474 Huang, Y. (2015). Pragmatics (Second ed.). Oxford Textbooks in Linguistics. Oxford
 11475 University Press.
- 11476 Huang, Z., W. Xu, and K. Yu (2015). Bidirectional lstm-crf models for sequence tagging.
 11477 arXiv preprint arXiv:1508.01991.
- 11478 Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes.
 11479 *Proceedings of the IRE* 40(9), 1098–1101.
- 11480 Humphreys, K., R. Gaizauskas, and S. Azzam (1997). Event coreference for information
 11481 extraction. In Proceedings of a Workshop on Operational Factors in Practical, Robust
 11482 Anaphora Resolution for Unrestricted Texts, pp. 75–81. Association for Computational
 11483 Linguistics.
- 11484 Ide, N. and Y. Wilks (2006). Making sense about sense. In Word sense disambiguation,
 11485 pp. 47–73. Springer.
- 11486 Ioffe, S. and C. Szegedy (2015). Batch normalization: Accelerating deep network training
 11487 by reducing internal covariate shift. In Proceedings of the International Conference on
 11488 Machine Learning (ICML), pp. 448–456.
- 11489 Isozaki, H., T. Hirao, K. Duh, K. Sudoh, and H. Tsukada (2010). Automatic evaluation of
 11490 translation quality for distant language pairs. In Proceedings of Empirical Methods for
 11491 Natural Language Processing (EMNLP), pp. 944–952.

- 11492 Iyyer, M., V. Manjunatha, J. Boyd-Graber, and H. Daumé III (2015). Deep unordered
11493 composition rivals syntactic methods for text classification. In Proceedings of the
11494 Association for Computational Linguistics (ACL), pp. 1681–1691.
- 11495 James, G., D. Witten, T. Hastie, and R. Tibshirani (2013). An introduction to statistical
11496 learning, Volume 112. Springer.
- 11497 Janin, A., D. Baron, J. Edwards, D. Ellis, D. Gelbart, N. Morgan, B. Peskin, T. Pfau,
11498 E. Shriberg, A. Stolcke, et al. (2003). The ICSI meeting corpus. In Acoustics, Speech, and
11499 Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference
11500 on, Volume 1, pp. I–I. IEEE.
- 11501 Jean, S., K. Cho, R. Memisevic, and Y. Bengio (2015). On using very large target
11502 vocabulary for neural machine translation. In Proceedings of the Association for
11503 Computational Linguistics (ACL), pp. 1–10.
- 11504 Jeong, M., C.-Y. Lin, and G. G. Lee (2009). Semi-supervised speech act recognition
11505 in emails and forums. In Proceedings of Empirical Methods for Natural Language
11506 Processing (EMNLP), pp. 1250–1259.
- 11507 Ji, H. and R. Grishman (2011). Knowledge base population: Successful approaches and
11508 challenges. In Proceedings of the Association for Computational Linguistics (ACL), pp.
11509 1148–1158.
- 11510 Ji, Y., T. Cohn, L. Kong, C. Dyer, and J. Eisenstein (2015). Document context language
11511 models. In International Conference on Learning Representations, Workshop Track,
11512 Volume abs/1511.03962.
- 11513 Ji, Y. and J. Eisenstein (2014). Representation learning for text-level discourse parsing. In
11514 Proceedings of the Association for Computational Linguistics (ACL).
- 11515 Ji, Y. and J. Eisenstein (2015, June). One vector is not enough: Entity-augmented
11516 distributional semantics for discourse relations. Transactions of the Association for
11517 Computational Linguistics (TACL).
- 11518 Ji, Y., G. Haffari, and J. Eisenstein (2016). A latent variable recurrent neural network for
11519 discourse relation language models. In Proceedings of the North American Chapter of
11520 the Association for Computational Linguistics (NAACL).
- 11521 Ji, Y. and N. A. Smith (2017). Neural discourse structure for text categorization. In
11522 Proceedings of the Association for Computational Linguistics (ACL), pp. 996–1005.
- 11523 Ji, Y., C. Tan, S. Martschat, Y. Choi, and N. A. Smith (2017). Dynamic entity
11524 representations in neural language models. In Proceedings of Empirical Methods for
11525 Natural Language Processing (EMNLP), pp. 1831–1840.

- 11526 Jiang, L., M. Yu, M. Zhou, X. Liu, and T. Zhao (2011). Target-dependent twitter sentiment
11527 classification. In Proceedings of the Association for Computational Linguistics (ACL),
11528 pp. 151–160.
- 11529 Jing, H. (2000). Sentence reduction for automatic text summarization. In Proceedings of
11530 the sixth conference on Applied natural language processing, pp. 310–315. Association
11531 for Computational Linguistics.
- 11532 Joachims, T. (2002). Optimizing search engines using clickthrough data. In Proceedings
11533 of Knowledge Discovery and Data Mining (KDD), pp. 133–142.
- 11534 Jockers, M. L. (2015). Szuzhet? <http://bla.bla.com>.
- 11535 Johnson, A. E., T. J. Pollard, L. Shen, H. L. Li-wei, M. Feng, M. Ghassemi, B. Moody,
11536 P. Szolovits, L. A. Celi, and R. G. Mark (2016). Mimic-iii, a freely accessible critical
11537 care database. *Scientific data* 3, 160035.
- 11538 Johnson, M. (1998). Pcfg models of linguistic tree representations. *Computational
11539 Linguistics* 24(4), 613–632.
- 11540 Johnson, R. and T. Zhang (2017). Deep pyramid convolutional neural networks for text
11541 categorization. In Proceedings of the Association for Computational Linguistics (ACL),
11542 pp. 562–570.
- 11543 Joshi, A. K. (1985). How much context-sensitivity is required to provide reasonable
11544 structural descriptions? – tree adjoining grammars. In *Natural Language Processing –
11545 Theoretical, Computational and Psychological Perspective*. New York, NY: Cambridge
11546 University Press.
- 11547 Joshi, A. K. and Y. Schabes (1997). Tree-adjoining grammars. In *Handbook of formal
11548 languages*, pp. 69–123. Springer.
- 11549 Joshi, A. K., K. V. Shanker, and D. Weir (1991). The convergence of mildly
11550 context-sensitive grammar formalisms. In *Foundational Issues in Natural Language
11551 Processing*. Cambridge MA: MIT Press.
- 11552 Jozefowicz, R., O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu (2016). Exploring the
11553 limits of language modeling. *arXiv preprint arXiv:1602.02410*.
- 11554 Jozefowicz, R., W. Zaremba, and I. Sutskever (2015). An empirical exploration of recurrent
11555 network architectures. In Proceedings of the International Conference on Machine
11556 Learning (ICML), pp. 2342–2350.
- 11557 Jurafsky, D. (1996). A probabilistic model of lexical and syntactic access and
11558 disambiguation. *Cognitive Science* 20(2), 137–194.

- 11559 Jurafsky, D. and J. H. Martin (2009). *Speech and Language Processing* (Second ed.).
11560 Prentice Hall.
- 11561 Jurafsky, D. and J. H. Martin (2018). *Speech and Language Processing* (Third ed.).
11562 Prentice Hall.
- 11563 Kadlec, R., M. Schmid, O. Bajgar, and J. Kleindienst (2016). Text understanding with
11564 the attention sum reader network. In Proceedings of the Association for Computational
11565 Linguistics (ACL), pp. 908–918.
- 11566 Kalchbrenner, N. and P. Blunsom (2013, August). Recurrent convolutional neural networks
11567 for discourse compositionality. In Proceedings of the Workshop on Continuous Vector
11568 Space Models and their Compositionality, Sofia, Bulgaria, pp. 119–126. Association for
11569 Computational Linguistics.
- 11570 Kalchbrenner, N., E. Grefenstette, and P. Blunsom (2014). A convolutional neural network
11571 for modelling sentences. In Proceedings of the Association for Computational Linguistics
11572 (ACL), pp. 655–665.
- 11573 Karlsson, F. (2007). Constraints on multiple center-embedding of clauses. *Journal of*
11574 *Linguistics* 43(02), 365–392.
- 11575 Kate, R. J., Y. W. Wong, and R. J. Mooney (2005). Learning to transform natural to
11576 formal languages. In Proceedings of the National Conference on Artificial Intelligence
11577 (AAAI).
- 11578 Kehler, A. (2007). Rethinking the smash approach to pronoun interpretation. In
11579 Interdisciplinary perspectives on reference processing, New Directions in Cognitive
11580 Science Series, pp. 95–122. Oxford University Press.
- 11581 Kibble, R. and R. Power (2004). Optimizing referential coherence in text generation.
11582 *Computational Linguistics* 30(4), 401–416.
- 11583 Kilgarriff, A. (1997). I don't believe in word senses. *Computers and the Humanities* 31(2),
11584 91–113.
- 11585 Kilgarriff, A. and G. Grefenstette (2003). Introduction to the special issue on the web as
11586 corpus. *Computational linguistics* 29(3), 333–347.
- 11587 Kim, M.-J. (2002). Does korean have adjectives? *MIT Working Papers in Linguistics* 43,
11588 71–89.
- 11589 Kim, S.-M. and E. Hovy (2006, July). Extracting opinions, opinion holders, and topics
11590 expressed in online news media text. In Proceedings of the Workshop on Sentiment
11591 and Subjectivity in Text, Sydney, Australia, pp. 1–8. Association for Computational
11592 Linguistics.

- 11593 Kim, Y. (2014). Convolutional neural networks for sentence classification. In Proceedings
 11594 of Empirical Methods for Natural Language Processing (EMNLP), pp. 1746–1751.
- 11595 Kim, Y., C. Denton, L. Hoang, and A. M. Rush (2017). Structured attention networks. In
 11596 Proceedings of the International Conference on Learning Representations (ICLR).
- 11597 Kim, Y., Y. Jernite, D. Sontag, and A. M. Rush (2016). Character-aware neural language
 11598 models. In Proceedings of the National Conference on Artificial Intelligence (AAAI).
- 11599 Kingma, D. and J. Ba (2014). Adam: A method for stochastic optimization. arXiv preprint
 11600 arXiv:1412.6980.
- 11601 Kiperwasser, E. and Y. Goldberg (2016). Simple and accurate dependency parsing
 11602 using bidirectional lstm feature representations. Transactions of the Association for
 11603 Computational Linguistics 4, 313–327.
- 11604 Kipper-Schuler, K. (2005). VerbNet: A broad-coverage, comprehensive verb lexicon. Ph.
 11605 D. thesis, Computer and Information Science, University of Pennsylvania.
- 11606 Kiros, R., R. Salakhutdinov, and R. Zemel (2014). Multimodal neural language models. In
 11607 Proceedings of the International Conference on Machine Learning (ICML), pp. 595–603.
- 11608 Kiros, R., Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler
 11609 (2015). Skip-thought vectors. In Neural Information Processing Systems (NIPS).
- 11610 Klein, D. and C. D. Manning (2003). Accurate unlexicalized parsing. In Proceedings of
 11611 the Association for Computational Linguistics (ACL), pp. 423–430.
- 11612 Klein, D. and C. D. Manning (2004). Corpus-based induction of syntactic structure: Models
 11613 of dependency and constituency. In Proceedings of the Association for Computational
 11614 Linguistics (ACL).
- 11615 Klein, G., Y. Kim, Y. Deng, J. Senellart, and A. M. Rush (2017). Opennmt: Open-source
 11616 toolkit for neural machine translation. arXiv preprint arXiv:1701.02810.
- 11617 Klementiev, A., I. Titov, and B. Bhattachari (2012). Inducing crosslingual distributed
 11618 representations of words. In Proceedings of the International Conference on
 11619 Computational Linguistics (COLING), pp. 1459–1474.
- 11620 Klenner, M. (2007). Enforcing consistency on coreference sets. In Recent Advances in
 11621 Natural Language Processing (RANLP), pp. 323–328.
- 11622 Knight, K. (1999). Decoding complexity in word-replacement translation models.
 11623 Computational Linguistics 25(4), 607–615.

- 11624 Knight, K. and J. Graehl (1998). Machine transliteration. *Computational Linguistics* 24(4),
11625 599–612.
- 11626 Knight, K. and D. Marcu (2000). Statistics-based summarization-step one: Sentence
11627 compression. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*,
11628 pp. 703–710.
- 11629 Knight, K. and J. May (2009). Applications of weighted automata in natural language
11630 processing. In *Handbook of Weighted Automata*, pp. 571–596. Springer.
- 11631 Knott, A. (1996). A data-driven methodology for motivating a set of coherence relations.
11632 Ph. D. thesis, The University of Edinburgh.
- 11633 Koehn, P. (2005). Europarl: A parallel corpus for statistical machine translation. In *MT*
11634 *summit*, Volume 5, pp. 79–86.
- 11635 Koehn, P. (2009). Statistical machine translation. Cambridge University Press.
- 11636 Koehn, P. (2017). Neural machine translation. arXiv preprint arXiv:1709.07809.
- 11637 Konstas, I. and M. Lapata (2013). A global model for concept-to-text generation. *Journal*
11638 *of Artificial Intelligence Research* 48, 305–346.
- 11639 Koo, T., X. Carreras, and M. Collins (2008, jun). Simple semi-supervised dependency
11640 parsing. In *Proceedings of ACL-08: HLT*, Columbus, Ohio, pp. 595–603. Association for
11641 Computational Linguistics.
- 11642 Koo, T. and M. Collins (2005). Hidden-variable models for discriminative reranking. In
11643 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 507–
11644 514.
- 11645 Koo, T. and M. Collins (2010). Efficient third-order dependency parsers. In *Proceedings*
11646 *of the Association for Computational Linguistics (ACL)*.
- 11647 Koo, T., A. Globerson, X. Carreras, and M. Collins (2007). Structured prediction models
11648 via the matrix-tree theorem. In *Proceedings of Empirical Methods for Natural Language*
11649 *Processing (EMNLP)*, pp. 141–150.
- 11650 Kovach, B. and T. Rosenstiel (2014). *The elements of journalism: What newspeople should*
11651 *know and the public should expect*. Three Rivers Press.
- 11652 Krishnamurthy, J. (2016). Probabilistic models for learning a semantic parser lexicon.
11653 In *Proceedings of the North American Chapter of the Association for Computational*
11654 *Linguistics (NAACL)*, pp. 606–616.

- 11655 Krishnamurthy, J. and T. M. Mitchell (2012). Weakly supervised training of semantic
 11656 parsers. In Proceedings of Empirical Methods for Natural Language Processing
 11657 (EMNLP), pp. 754–765.
- 11658 Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep
 11659 convolutional neural networks. In Neural Information Processing Systems (NIPS), pp.
 11660 1097–1105.
- 11661 Kübler, S., R. McDonald, and J. Nivre (2009). Dependency parsing. Synthesis Lectures
 11662 on Human Language Technologies 1(1), 1–127.
- 11663 Kuhlmann, M. and J. Nivre (2010). Transition-based techniques for non-projective
 11664 dependency parsing. Northern European Journal of Language Technology (NEJLT) 2(1),
 11665 1–19.
- 11666 Kummerfeld, J. K., T. Berg-Kirkpatrick, and D. Klein (2015). An empirical analysis of
 11667 optimization for max-margin NLP. In Proceedings of Empirical Methods for Natural
 11668 Language Processing (EMNLP).
- 11669 Kwiatkowski, T., S. Goldwater, L. Zettlemoyer, and M. Steedman (2012). A probabilistic
 11670 model of syntactic and semantic acquisition from child-directed utterances and their
 11671 meanings. In Proceedings of the European Chapter of the Association for Computational
 11672 Linguistics (EACL), pp. 234–244.
- 11673 Lafferty, J., A. McCallum, and F. Pereira (2001). Conditional random fields: Probabilistic
 11674 models for segmenting and labeling sequence data. In icml.
- 11675 Lakoff, G. (1973). Hedges: A study in meaning criteria and the logic of fuzzy concepts.
 11676 Journal of philosophical logic 2(4), 458–508.
- 11677 Lample, G., M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer (2016). Neural
 11678 architectures for named entity recognition. In Proceedings of the North American
 11679 Chapter of the Association for Computational Linguistics (NAACL), pp. 260–270.
- 11680 Langkilde, I. and K. Knight (1998). Generation that exploits corpus-based statistical
 11681 knowledge. In Proceedings of the Association for Computational Linguistics (ACL), pp.
 11682 704–710.
- 11683 Lapata, M. (2003). Probabilistic text structuring: Experiments with sentence ordering. In
 11684 Proceedings of the Association for Computational Linguistics (ACL), pp. 545–552.
- 11685 Lappin, S. and H. J. Leass (1994). An algorithm for pronominal anaphora resolution.
 11686 Computational linguistics 20(4), 535–561.

- 11687 Lari, K. and S. J. Young (1990). The estimation of stochastic context-free grammars using
11688 the inside-outside algorithm. *Computer speech & language* 4(1), 35–56.
- 11689 Lascarides, A. and N. Asher (2007). Segmented discourse representation theory: Dynamic
11690 semantics with discourse structure. In *Computing meaning*, pp. 87–124. Springer.
- 11691 Law, E. and L. v. Ahn (2011). Human computation. *Synthesis Lectures on Artificial
11692 Intelligence and Machine Learning* 5(3), 1–121.
- 11693 Lebret, R., D. Grangier, and M. Auli (2016). Neural text generation from structured data
11694 with application to the biography domain. In *Proceedings of Empirical Methods for
11695 Natural Language Processing (EMNLP)*, pp. 1203–1213.
- 11696 LeCun, Y. and Y. Bengio (1995). Convolutional networks for images, speech, and time
11697 series. *The handbook of brain theory and neural networks* 3361.
- 11698 LeCun, Y., L. Bottou, G. B. Orr, and K.-R. Müller (1998). Efficient backprop. In *Neural
11699 networks: Tricks of the trade*, pp. 9–50. Springer.
- 11700 Lee, C. M. and S. S. Narayanan (2005). Toward detecting emotions in spoken dialogs.
11701 *IEEE transactions on speech and audio processing* 13(2), 293–303.
- 11702 Lee, H., A. Chang, Y. Peirsman, N. Chambers, M. Surdeanu, and D. Jurafsky (2013).
11703 Deterministic coreference resolution based on entity-centric, precision-ranked rules.
11704 *Computational Linguistics* 39(4), 885–916.
- 11705 Lee, H., Y. Peirsman, A. Chang, N. Chambers, M. Surdeanu, and D. Jurafsky (2011).
11706 Stanford’s multi-pass sieve coreference resolution system at the conll-2011 shared task.
11707 In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 28–34.
11708 Association for Computational Linguistics.
- 11709 Lee, K., L. He, M. Lewis, and L. Zettlemoyer (2017). End-to-end neural coreference
11710 resolution. In *Proceedings of Empirical Methods for Natural Language Processing
11711 (EMNLP)*.
- 11712 Lenat, D. B., R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd (1990). Cyc: toward
11713 programs with common sense. *Communications of the ACM* 33(8), 30–49.
- 11714 Lesk, M. (1986). Automatic sense disambiguation using machine readable dictionaries: how
11715 to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual international
11716 conference on Systems documentation*, pp. 24–26. ACM.
- 11717 Levesque, H. J., E. Davis, and L. Morgenstern (2011). The winograd schema challenge. In
11718 *Aaa spring symposium: Logical formalizations of commonsense reasoning*, Volume 46,
11719 pp. 47.

- 11720 Levin, E., R. Pieraccini, and W. Eckert (1998). Using markov decision process for learning
11721 dialogue strategies. In Acoustics, Speech and Signal Processing, 1998. Proceedings of
11722 the 1998 IEEE International Conference on, Volume 1, pp. 201–204. IEEE.
- 11723 Levy, O. and Y. Goldberg (2014). Dependency-based word embeddings. In Proceedings of
11724 the Association for Computational Linguistics (ACL), pp. 302–308.
- 11725 Levy, O., Y. Goldberg, and I. Dagan (2015). Improving distributional similarity
11726 with lessons learned from word embeddings. Transactions of the Association for
11727 Computational Linguistics 3, 211–225.
- 11728 Levy, R. and C. Manning (2009). An informal introduction to computational semantics.
- 11729 Lewis, M. and M. Steedman (2013). Combined distributional and logical semantics.
11730 Transactions of the Association for Computational Linguistics 1, 179–192.
- 11731 Lewis II, P. M. and R. E. Stearns (1968). Syntax-directed transduction. Journal of the
11732 ACM (JACM) 15(3), 465–488.
- 11733 Li, J. and D. Jurafsky (2015). Do multi-sense embeddings improve natural language
11734 understanding? In Proceedings of Empirical Methods for Natural Language Processing
11735 (EMNLP), pp. 1722–1732.
- 11736 Li, J. and D. Jurafsky (2017). Neural net models of open-domain discourse coherence.
11737 In Proceedings of Empirical Methods for Natural Language Processing (EMNLP), pp.
11738 198–209.
- 11739 Li, J., R. Li, and E. Hovy (2014). Recursive deep models for discourse parsing. In
11740 Proceedings of Empirical Methods for Natural Language Processing (EMNLP).
- 11741 Li, J., M.-T. Luong, and D. Jurafsky (2015). A hierarchical neural autoencoder for
11742 paragraphs and documents. In Proceedings of Empirical Methods for Natural Language
11743 Processing (EMNLP).
- 11744 Li, J., T. Luong, D. Jurafsky, and E. Hovy (2015). When are tree structures necessary
11745 for deep learning of representations? In Proceedings of Empirical Methods for Natural
11746 Language Processing (EMNLP), pp. 2304–2314.
- 11747 Li, J., W. Monroe, A. Ritter, D. Jurafsky, M. Galley, and J. Gao (2016, November). Deep
11748 reinforcement learning for dialogue generation. In Proceedings of the 2016 Conference
11749 on Empirical Methods in Natural Language Processing, Austin, Texas, pp. 1192–1202.
11750 Association for Computational Linguistics.
- 11751 Li, Q., S. Anzaroot, W.-P. Lin, X. Li, and H. Ji (2011). Joint inference for cross-document
11752 information extraction. In Proceedings of the International Conference on Information
11753 and Knowledge Management (CIKM), pp. 2225–2228.

- 11754 Li, Q., H. Ji, and L. Huang (2013). Joint event extraction via structured prediction with
11755 global features. In Proceedings of the Association for Computational Linguistics (ACL),
11756 pp. 73–82.
- 11757 Liang, P., A. Bouchard-Côté, D. Klein, and B. Taskar (2006). An end-to-end discriminative
11758 approach to machine translation. In Proceedings of the Association for Computational
11759 Linguistics (ACL), pp. 761–768.
- 11760 Liang, P., M. Jordan, and D. Klein (2009). Learning semantic correspondences with less
11761 supervision. In Proceedings of the Association for Computational Linguistics (ACL), pp.
11762 91–99.
- 11763 Liang, P., M. I. Jordan, and D. Klein (2013). Learning dependency-based compositional
11764 semantics. *Computational Linguistics* 39(2), 389–446.
- 11765 Liang, P. and D. Klein (2009). Online em for unsupervised models. In Proceedings of the
11766 North American Chapter of the Association for Computational Linguistics (NAACL),
11767 pp. 611–619.
- 11768 Liang, P., S. Petrov, M. I. Jordan, and D. Klein (2007). The infinite pcfg using
11769 hierarchical dirichlet processes. In Proceedings of Empirical Methods for Natural
11770 Language Processing (EMNLP), pp. 688–697.
- 11771 Liang, P. and C. Potts (2015). Bringing machine learning and compositional semantics
11772 together. *Annual Review of Linguistics* 1(1), 355–376.
- 11773 Lieber, R. (2015). Introducing morphology. Cambridge University Press.
- 11774 Lin, D. (1998). Automatic retrieval and clustering of similar words. In Proceedings of
11775 the 17th international conference on Computational linguistics-Volume 2, pp. 768–774.
11776 Association for Computational Linguistics.
- 11777 Lin, J. and C. Dyer (2010). Data-intensive text processing with mapreduce. *Synthesis*
11778 Lectures on Human Language Technologies 3(1), 1–177.
- 11779 Lin, Z., M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio (2017). A
11780 structured self-attentive sentence embedding. arXiv preprint arXiv:1703.03130.
- 11781 Lin, Z., M.-Y. Kan, and H. T. Ng (2009). Recognizing implicit discourse relations in the
11782 penn discourse treebank. In Proceedings of Empirical Methods for Natural Language
11783 Processing (EMNLP), pp. 343–351.
- 11784 Lin, Z., H. T. Ng, and M.-Y. Kan (2011). Automatically evaluating text coherence using
11785 discourse relations. In Proceedings of the Association for Computational Linguistics
11786 (ACL), pp. 997–1006.

- 11787 Lin, Z., H. T. Ng, and M. Y. Kan (2014, nov). A PDTB-styled end-to-end discourse parser.
11788 Natural Language Engineering FirstView, 1–34.
- 11789 Ling, W., C. Dyer, A. Black, and I. Trancoso (2015). Two/too simple adaptations of
11790 word2vec for syntax problems. In Proceedings of the North American Chapter of the
11791 Association for Computational Linguistics (NAACL).
- 11792 Ling, W., T. Luís, L. Marujo, R. F. Astudillo, S. Amir, C. Dyer, A. W. Black, and
11793 I. Trancoso (2015). Finding function in form: Compositional character models for
11794 open vocabulary word representation. In Proceedings of Empirical Methods for Natural
11795 Language Processing (EMNLP).
- 11796 Ling, W., G. Xiang, C. Dyer, A. Black, and I. Trancoso (2013). Microblogs as parallel
11797 corpora. In Proceedings of the Association for Computational Linguistics (ACL).
- 11798 Ling, X., S. Singh, and D. S. Weld (2015). Design challenges for entity linking. Transactions
11799 of the Association for Computational Linguistics 3, 315–328.
- 11800 Linguistic Data Consortium (2005, July). ACE (automatic content extraction) English
11801 annotation guidelines for relations. Technical Report Version 5.8.3, Linguistic Data
11802 Consortium.
- 11803 Liu, B. (2015). Sentiment Analysis: Mining Opinions, Sentiments, and Emotions.
11804 Cambridge University Press.
- 11805 Liu, D. C. and J. Nocedal (1989). On the limited memory BFGS method for large scale
11806 optimization. Mathematical programming 45(1-3), 503–528.
- 11807 Liu, Y., Q. Liu, and S. Lin (2006). Tree-to-string alignment template for statistical machine
11808 translation. In Proceedings of the Association for Computational Linguistics (ACL), pp.
11809 609–616.
- 11810 Loper, E. and S. Bird (2002). Nltk: The natural language toolkit. In Proceedings
11811 of the ACL-02 Workshop on Effective tools and methodologies for teaching natural
11812 language processing and computational linguistics-Volume 1, pp. 63–70. Association for
11813 Computational Linguistics.
- 11814 Louis, A., A. Joshi, and A. Nenkova (2010). Discourse indicators for content selection in
11815 summarization. In Proceedings of the 11th Annual Meeting of the Special Interest Group
11816 on Discourse and Dialogue, pp. 147–156. Association for Computational Linguistics.
- 11817 Louis, A. and A. Nenkova (2013). What makes writing great? first experiments on article
11818 quality prediction in the science journalism domain. Transactions of the Association for
11819 Computational Linguistics 1, 341–352.

- 11820 Loveland, D. W. (2016). Automated Theorem Proving: a logical basis. Elsevier.
- 11821 Lowe, R., N. Pow, I. V. Serban, and J. Pineau (2015). The ubuntu dialogue corpus: A
11822 large dataset for research in unstructured multi-turn dialogue systems. In Proceedings
11823 of the Special Interest Group on Discourse and Dialogue (SIGDIAL).
- 11824 Luo, X. (2005). On coreference resolution performance metrics. In Proceedings of Empirical
11825 Methods for Natural Language Processing (EMNLP), pp. 25–32.
- 11826 Luo, X., A. Ittycheriah, H. Jing, N. Kambhatla, and S. Roukos (2004). A
11827 mention-synchronous coreference resolution algorithm based on the bell tree. In
11828 Proceedings of the Association for Computational Linguistics (ACL).
- 11829 Luong, M.-T., R. Socher, and C. D. Manning (2013). Better word representations with
11830 recursive neural networks for morphology. CoNLL-2013 104.
- 11831 Luong, T., H. Pham, and C. D. Manning (2015). Effective approaches to attention-based
11832 neural machine translation. In Proceedings of Empirical Methods for Natural Language
11833 Processing (EMNLP), pp. 1412–1421.
- 11834 Luong, T., I. Sutskever, Q. Le, O. Vinyals, and W. Zaremba (2015). Addressing the
11835 rare word problem in neural machine translation. In Proceedings of the Association for
11836 Computational Linguistics (ACL), pp. 11–19.
- 11837 Maas, A. L., A. Y. Hannun, and A. Y. Ng (2013). Rectifier nonlinearities improve neural
11838 network acoustic models. In Proceedings of the International Conference on Machine
11839 Learning (ICML).
- 11840 Mairesse, F. and M. A. Walker (2011). Controlling user perceptions of linguistic style:
11841 Trainable generation of personality traits. Computational Linguistics 37(3), 455–488.
- 11842 Mani, I., M. Verhagen, B. Wellner, C. M. Lee, and J. Pustejovsky (2006). Machine learning
11843 of temporal relations. In Proceedings of the Association for Computational Linguistics
11844 (ACL), pp. 753–760.
- 11845 Mann, W. C. and S. A. Thompson (1988). Rhetorical structure theory: Toward a functional
11846 theory of text organization. Text 8(3), 243–281.
- 11847 Manning, C. D. (2015). Computational linguistics and deep learning. Computational
11848 Linguistics 41(4), 701–707.
- 11849 Manning, C. D. (2016). Computational linguistics and deep learning. Computational
11850 Linguistics 41(4).
- 11851 Manning, C. D., P. Raghavan, H. Schütze, et al. (2008). Introduction to information
11852 retrieval, Volume 1. Cambridge university press.

- 11853 Manning, C. D. and H. Schütze (1999). Foundations of Statistical Natural Language
11854 Processing. Cambridge, Massachusetts: MIT press.
- 11855 Marcu, D. (1996). Building up rhetorical structure trees. In Proceedings of the National
11856 Conference on Artificial Intelligence, pp. 1069–1074.
- 11857 Marcu, D. (1997a). From discourse structures to text summaries. In Proceedings of the
11858 workshop on Intelligent Scalable Text Summarization.
- 11859 Marcu, D. (1997b). From local to global coherence: A bottom-up approach to text planning.
11860 In Proceedings of the National Conference on Artificial Intelligence (AAAI), pp. 629–635.
- 11861 Marcus, M. P., M. A. Marcinkiewicz, and B. Santorini (1993). Building a large annotated
11862 corpus of English: The Penn Treebank. Computational Linguistics 19(2), 313–330.
- 11863 Maron, O. and T. Lozano-Pérez (1998). A framework for multiple-instance learning. In
11864 Neural Information Processing Systems (NIPS), pp. 570–576.
- 11865 Márquez, G. G. (1970). One Hundred Years of Solitude. Harper & Row. English translation
11866 by Gregory Rabassa.
- 11867 Martins, A. F. T., N. A. Smith, and E. P. Xing (2009). Concise integer linear
11868 programming formulations for dependency parsing. In Proceedings of the Association
11869 for Computational Linguistics (ACL), pp. 342–350.
- 11870 Martins, A. F. T., N. A. Smith, E. P. Xing, P. M. Q. Aguiar, and M. A. T. Figueiredo
11871 (2010). Turbo parsers: Dependency parsing by approximate variational inference. In
11872 Proceedings of Empirical Methods for Natural Language Processing (EMNLP), pp. 34–
11873 44.
- 11874 Matsuzaki, T., Y. Miyao, and J. Tsuji (2005). Probabilistic cfg with latent annotations.
11875 In Proceedings of the Association for Computational Linguistics (ACL), pp. 75–82.
- 11876 Matthiessen, C. and J. A. Bateman (1991). Text generation and systemic-functional
11877 linguistics: experiences from English and Japanese. Pinter Publishers.
- 11878 McCallum, A. and W. Li (2003). Early results for named entity recognition with conditional
11879 random fields, feature induction and web-enhanced lexicons. In Proceedings of the North
11880 American Chapter of the Association for Computational Linguistics (NAACL), pp. 188–
11881 191.
- 11882 McCallum, A. and B. Wellner (2004). Conditional models of identity uncertainty with
11883 application to noun coreference. In NIPS, pp. 905–912.

- 11884 McDonald, R., K. Crammer, and F. Pereira (2005). Online large-margin training of
11885 dependency parsers. In Proceedings of the Association for Computational Linguistics
11886 (ACL), pp. 91–98.
- 11887 McDonald, R., K. Hannan, T. Neylon, M. Wells, and J. Reynar (2007). Structured models
11888 for fine-to-coarse sentiment analysis. In Proceedings of ACL.
- 11889 McDonald, R. and F. Pereira (2006). Online learning of approximate dependency
11890 parsing algorithms. In Proceedings of the European Chapter of the Association for
11891 Computational Linguistics (EACL).
- 11892 McKeown, K. (1992). Text generation. Cambridge University Press.
- 11893 McKeown, K., S. Rosenthal, K. Thadani, and C. Moore (2010). Time-efficient creation of
11894 an accurate sentence fusion corpus. In Proceedings of the North American Chapter of
11895 the Association for Computational Linguistics (NAACL), pp. 317–320.
- 11896 McKeown, K. R., R. Barzilay, D. Evans, V. Hatzivassiloglou, J. L. Klavans, A. Nenkova,
11897 C. Sable, B. Schiffman, and S. Sigelman (2002). Tracking and summarizing news on
11898 a daily basis with columbia’s newsblaster. In Proceedings of the second international
11899 conference on Human Language Technology Research, pp. 280–285.
- 11900 McNamee, P. and H. T. Dang (2009). Overview of the tac 2009 knowledge base population
11901 track. In Text Analysis Conference (TAC), Volume 17, pp. 111–113.
- 11902 Medlock, B. and T. Briscoe (2007). Weakly supervised learning for hedge classification
11903 in scientific literature. In Proceedings of the Association for Computational Linguistics
11904 (ACL), pp. 992–999.
- 11905 Mei, H., M. Bansal, and M. R. Walter (2016). What to talk about and how? selective
11906 generation using lstms with coarse-to-fine alignment. In Proceedings of the North
11907 American Chapter of the Association for Computational Linguistics (NAACL), pp. 720–
11908 730.
- 11909 Merity, S., N. S. Keskar, and R. Socher (2018). Regularizing and optimizing lstm language
11910 models. In Proceedings of the International Conference on Learning Representations
11911 (ICLR).
- 11912 Merity, S., C. Xiong, J. Bradbury, and R. Socher (2017). Pointer sentinel mixture models.
11913 In Proceedings of the International Conference on Learning Representations (ICLR).
- 11914 Messud, C. (2014, June). A new ‘l’étranger’. New York Review of Books.
- 11915 Miao, Y. and P. Blunsom (2016). Language as a latent variable: Discrete generative models
11916 for sentence compression. In Proceedings of Empirical Methods for Natural Language
11917 Processing (EMNLP), pp. 319–328.

- 11918 Miao, Y., L. Yu, and P. Blunsom (2016). Neural variational inference for text processing.
 11919 In Proceedings of the International Conference on Machine Learning (ICML).
- 11920 Mihalcea, R., T. A. Chklovski, and A. Kilgarriff (2004, July). The senseval-3 english lexical
 11921 sample task. In Proceedings of SENSEVAL-3, Barcelona, Spain, pp. 25–28. Association
 11922 for Computational Linguistics.
- 11923 Mihalcea, R. and D. Radev (2011). Graph-based natural language processing and
 11924 information retrieval. Cambridge University Press.
- 11925 Mikolov, T., K. Chen, G. Corrado, and J. Dean (2013). Efficient estimation of word
 11926 representations in vector space. In Proceedings of International Conference on Learning
 11927 Representations.
- 11928 Mikolov, T., A. Deoras, D. Povey, L. Burget, and J. Cernocky (2011). Strategies for
 11929 training large scale neural network language models. In Automatic Speech Recognition
 11930 and Understanding (ASRU), 2011 IEEE Workshop on, pp. 196–201. IEEE.
- 11931 Mikolov, T., M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur (2010). Recurrent
 11932 neural network based language model. In INTERSPEECH, pp. 1045–1048.
- 11933 Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013). Distributed
 11934 representations of words and phrases and their compositionality. In Advances in Neural
 11935 Information Processing Systems, pp. 3111–3119.
- 11936 Mikolov, T., W.-t. Yih, and G. Zweig (2013). Linguistic regularities in continuous space
 11937 word representations. In Proceedings of the North American Chapter of the Association
 11938 for Computational Linguistics (NAACL), pp. 746–751.
- 11939 Mikolov, T. and G. Zweig. Context dependent recurrent neural network language model.
 11940 In Proceedings of Spoken Language Technology (SLT), pp. 234–239.
- 11941 Miller, G. A., G. A. Heise, and W. Lichten (1951). The intelligibility of speech as a function
 11942 of the context of the test materials. *Journal of experimental psychology* 41(5), 329.
- 11943 Miller, M., C. Sathi, D. Wiesenthal, J. Leskovec, and C. Potts (2011). Sentiment flow
 11944 through hyperlink networks. In Proceedings of the International Conference on Web and
 11945 Social Media (ICWSM).
- 11946 Miller, S., J. Guinness, and A. Zamanian (2004). Name tagging with word clusters and
 11947 discriminative training. In Proceedings of the North American Chapter of the Association
 11948 for Computational Linguistics (NAACL), pp. 337–342.
- 11949 Milne, D. and I. H. Witten (2008). Learning to link with wikipedia. In Proceedings of
 11950 the International Conference on Information and Knowledge Management (CIKM), pp.
 11951 509–518. ACM.

- 11952 Miltsakaki, E. and K. Kukich (2004). Evaluation of text coherence for electronic essay
11953 scoring systems. *Natural Language Engineering* 10(1), 25–55.
- 11954 Minka, T. P. (1999). From hidden markov models to linear dynamical systems. Tech. Rep.
11955 531, Vision and Modeling Group of Media Lab, MIT.
- 11956 Minsky, M. (1974). A framework for representing knowledge. Technical Report 306, MIT
11957 AI Laboratory.
- 11958 Minsky, M. and S. Papert (1969). *Perceptrons*. MIT press.
- 11959 Mintz, M., S. Bills, R. Snow, and D. Jurafsky (2009). Distant supervision for relation
11960 extraction without labeled data. In *Proceedings of the Association for Computational
11961 Linguistics (ACL)*, pp. 1003–1011.
- 11962 Mirza, P., R. Sprugnoli, S. Tonelli, and M. Speranza (2014). Annotating causality in
11963 the tempeval-3 corpus. In *Proceedings of the EACL 2014 Workshop on Computational
11964 Approaches to Causality in Language (CAtoCL)*, pp. 10–19.
- 11965 Misra, D. K. and Y. Artzi (2016). Neural shift-reduce ccg semantic parsing. In *Proceedings
11966 of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11967 Mitchell, J. and M. Lapata (2010). Composition in distributional models of semantics.
11968 *Cognitive Science* 34(8), 1388–1429.
- 11969 Miwa, M. and M. Bansal (2016). End-to-end relation extraction using lstms on sequences
11970 and tree structures. In *Proceedings of the Association for Computational Linguistics
11971 (ACL)*, pp. 1105–1116.
- 11972 Mnih, A. and G. Hinton (2007). Three new graphical models for statistical language
11973 modelling. In *Proceedings of the 24th international conference on Machine learning,
11974 ICML '07*, New York, NY, USA, pp. 641–648. ACM.
- 11975 Mnih, A. and G. E. Hinton (2008). A scalable hierarchical distributed language model. In
11976 *Neural Information Processing Systems (NIPS)*, pp. 1081–1088.
- 11977 Mnih, A. and Y. W. Teh (2012). A fast and simple algorithm for training neural
11978 probabilistic language models. In *Proceedings of the International Conference on
11979 Machine Learning (ICML)*.
- 11980 Mohammad, S. M. and P. D. Turney (2013). Crowdsourcing a word–emotion association
11981 lexicon. *Computational Intelligence* 29(3), 436–465.
- 11982 Mohri, M., F. Pereira, and M. Riley (2002). Weighted finite-state transducers in speech
11983 recognition. *Computer Speech & Language* 16(1), 69–88.

- 11984 Mohri, M., A. Rostamizadeh, and A. Talwalkar (2012). Foundations of machine learning.
 11985 MIT press.
- 11986 Montague, R. (1973). The proper treatment of quantification in ordinary english. In
 11987 Approaches to natural language, pp. 221–242. Springer.
- 11988 Moore, J. D. and C. L. Paris (1993, dec). Planning text for advisory dialogues: Capturing
 11989 intentional and rhetorical information. *Comput. Linguist.* 19(4), 651–694.
- 11990 Morante, R. and E. Blanco (2012). *sem 2012 shared task: Resolving the scope and focus
 11991 of negation. In Proceedings of the First Joint Conference on Lexical and Computational
 11992 Semantics-Volume 1: Proceedings of the main conference and the shared task, and
 11993 Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation,
 11994 pp. 265–274. Association for Computational Linguistics.
- 11995 Morante, R. and W. Daelemans (2009). Learning the scope of hedge cues in biomedical
 11996 texts. In Proceedings of the Workshop on Current Trends in Biomedical Natural
 11997 Language Processing, pp. 28–36. Association for Computational Linguistics.
- 11998 Morante, R. and C. Sporleder (2012). Modality and negation: An introduction to the
 11999 special issue. *Computational linguistics* 38(2), 223–260.
- 12000 Mostafazadeh, N., A. Grelish, N. Chambers, J. Allen, and L. Vanderwende (2016, June).
 12001 Caters: Causal and temporal relation scheme for semantic annotation of event structures.
 12002 In Proceedings of the Fourth Workshop on Events, San Diego, California, pp. 51–61.
 12003 Association for Computational Linguistics.
- 12004 Mueller, T., H. Schmid, and H. Schütze (2013). Efficient higher-order CRFs for
 12005 morphological tagging. In Proceedings of Empirical Methods for Natural Language
 12006 Processing (EMNLP), pp. 322–332.
- 12007 Müller, C. and M. Strube (2006). Multi-level annotation of linguistic data with mmax2.
 12008 Corpus technology and language pedagogy: New resources, new tools, new methods 3,
 12009 197–214.
- 12010 Muralidharan, A. and M. A. Hearst (2013). Supporting exploratory text analysis in
 12011 literature study. *Literary and linguistic computing* 28(2), 283–295.
- 12012 Murphy, K. P. (2012). Machine Learning: A Probabilistic Perspective. The MIT Press.
- 12013 Nakagawa, T., K. Inui, and S. Kurohashi (2010). Dependency tree-based sentiment
 12014 classification using crfs with hidden variables. In Proceedings of the North American
 12015 Chapter of the Association for Computational Linguistics (NAACL), pp. 786–794.

- 12016 Nakazawa, T., M. Yaguchi, K. Uchimoto, M. Utiyama, E. Sumita, S. Kurohashi, and
12017 H. Isahara (2016). ASPEC: Asian scientific paper excerpt corpus. In Proceedings of the
12018 Language Resources and Evaluation Conference, pp. 2204–2208.
- 12019 Navigli, R. (2009). Word sense disambiguation: A survey. ACM Computing Surveys
12020 (CSUR) 41(2), 10.
- 12021 Neal, R. M. and G. E. Hinton (1998). A view of the em algorithm that justifies incremental,
12022 sparse, and other variants. In Learning in graphical models, pp. 355–368. Springer.
- 12023 Nenkova, A. and K. McKeown (2012). A survey of text summarization techniques. In
12024 Mining text data, pp. 43–76. Springer.
- 12025 Neubig, G. (2017). Neural machine translation and sequence-to-sequence models: A
12026 tutorial. arXiv preprint arXiv:1703.01619.
- 12027 Neubig, G., C. Dyer, Y. Goldberg, A. Matthews, W. Ammar, A. Anastasopoulos,
12028 M. Ballesteros, D. Chiang, D. Clothiaux, T. Cohn, K. Duh, M. Faruqui, C. Gan,
12029 D. Garrette, Y. Ji, L. Kong, A. Kuncoro, G. Kumar, C. Malaviya, P. Michel, Y. Oda,
12030 M. Richardson, N. Saphra, S. Swayamdipta, and P. Yin (2017). Dynet: The dynamic
12031 neural network toolkit.
- 12032 Neubig, G., Y. Goldberg, and C. Dyer (2017). On-the-fly operation batching in dynamic
12033 computation graphs. In Neural Information Processing Systems (NIPS).
- 12034 Neuhaus, P. and N. Bröker (1997). The complexity of recognition of linguistically adequate
12035 dependency grammars. In eacl, pp. 337–343.
- 12036 Newman, D., C. Chemudugunta, and P. Smyth (2006). Statistical entity-topic models. In
12037 Proceedings of Knowledge Discovery and Data Mining (KDD), pp. 680–686.
- 12038 Ng, V. (2010). Supervised noun phrase coreference research: The first fifteen years. In
12039 Proceedings of the 48th annual meeting of the association for computational linguistics,
12040 pp. 1396–1411. Association for Computational Linguistics.
- 12041 Nguyen, D. and A. S. Dogruöz (2013). Word level language identification in online
12042 multilingual communication. In Proceedings of Empirical Methods for Natural Language
12043 Processing (EMNLP).
- 12044 Nguyen, D. T. and S. Joty (2017). A neural local coherence model. In Proceedings of the
12045 Association for Computational Linguistics (ACL), pp. 1320–1330.
- 12046 Nigam, K., A. K. McCallum, S. Thrun, and T. Mitchell (2000). Text classification from
12047 labeled and unlabeled documents using em. Machine learning 39(2-3), 103–134.

- 12048 Nirenburg, S. and Y. Wilks (2001). What's in a symbol: ontology, representation and
 12049 language. *Journal of Experimental & Theoretical Artificial Intelligence* 13(1), 9–23.
- 12050 Nivre, J. (2008). Algorithms for deterministic incremental dependency parsing.
 12051 *Computational Linguistics* 34(4), 513–553.
- 12052 Nivre, J., M.-C. de Marneffe, F. Ginter, Y. Goldberg, J. Hajič, C. D. Manning,
 12053 R. McDonald, S. Petrov, S. Pyysalo, N. Silveira, R. Tsarfaty, and D. Zeman (2016,
 12054 may). Universal dependencies v1: A multilingual treebank collection. In N. C. C.
 12055 Chair), K. Choukri, T. Declerck, S. Goggi, M. Grobelnik, B. Maegaard, J. Mariani,
 12056 H. Mazo, A. Moreno, J. Odijk, and S. Piperidis (Eds.), *Proceedings of the Tenth
 12057 International Conference on Language Resources and Evaluation (LREC 2016)*, Paris,
 12058 France. European Language Resources Association (ELRA).
- 12059 Nivre, J. and J. Nilsson (2005). Pseudo-projective dependency parsing. In *Proceedings
 12060 of the 43rd Annual Meeting on Association for Computational Linguistics*, pp. 99–106.
 12061 Association for Computational Linguistics.
- 12062 Novikoff, A. B. (1962). On convergence proofs on perceptrons. In *Proceedings of the
 12063 Symposium on the Mathematical Theory of Automata*, Volume 12, pp. 615–622.
- 12064 Och, F. J. and H. Ney (2003). A systematic comparison of various statistical alignment
 12065 models. *Computational linguistics* 29(1), 19–51.
- 12066 O'Connor, B., M. Krieger, and D. Ahn (2010). Tweetmotif: Exploratory search and topic
 12067 summarization for twitter. In *Proceedings of the International Conference on Web and
 12068 Social Media (ICWSM)*, pp. 384–385.
- 12069 Oflazer, K. and İ. Kuruöz (1994). Tagging and morphological disambiguation of turkish
 12070 text. In *Proceedings of the fourth conference on Applied natural language processing*,
 12071 pp. 144–149. Association for Computational Linguistics.
- 12072 Ohta, T., Y. Tateisi, and J.-D. Kim (2002). The genia corpus: An annotated research
 12073 abstract corpus in molecular biology domain. In *Proceedings of the second international
 12074 conference on Human Language Technology Research*, pp. 82–86. Morgan Kaufmann
 12075 Publishers Inc.
- 12076 Onishi, T., H. Wang, M. Bansal, K. Gimpel, and D. McAllester (2016). Who did what:
 12077 A large-scale person-centered cloze dataset. In *Proceedings of Empirical Methods for
 12078 Natural Language Processing (EMNLP)*, pp. 2230–2235.
- 12079 Owoputi, O., B. O'Connor, C. Dyer, K. Gimpel, N. Schneider, and N. A. Smith (2013).
 12080 Improved part-of-speech tagging for online conversational text with word clusters. In
 12081 *Proceedings of the North American Chapter of the Association for Computational
 12082 Linguistics (NAACL)*, pp. 380–390.

- 12083 Packard, W., E. M. Bender, J. Read, S. Oepen, and R. Dridan (2014). Simple negation
12084 scope resolution through deep parsing: A semantic solution to a semantic problem. In
12085 Proceedings of the Association for Computational Linguistics (ACL), pp. 69–78.
- 12086 Paice, C. D. (1990). Another stemmer. In ACM SIGIR Forum, Volume 24, pp. 56–61.
- 12087 Pak, A. and P. Paroubek (2010). Twitter as a corpus for sentiment analysis and opinion
12088 mining. In LREC, Volume 10, pp. 1320–1326.
- 12089 Palmer, F. R. (2001). Mood and modality. Cambridge University Press.
- 12090 Palmer, M., D. Gildea, and P. Kingsbury (2005). The proposition bank: An annotated
12091 corpus of semantic roles. Computational linguistics 31(1), 71–106.
- 12092 Pan, S. J. and Q. Yang (2010). A survey on transfer learning. IEEE Transactions on
12093 knowledge and data engineering 22(10), 1345–1359.
- 12094 Pan, X., T. Cassidy, U. Hermjakob, H. Ji, and K. Knight (2015). Unsupervised entity
12095 linking with abstract meaning representation. In Proceedings of the North American
12096 Chapter of the Association for Computational Linguistics (NAACL), pp. 1130–1139.
- 12097 Pang, B. and L. Lee (2004). A sentimental education: Sentiment analysis using
12098 subjectivity summarization based on minimum cuts. In Proceedings of the Association
12099 for Computational Linguistics (ACL), pp. 271–278.
- 12100 Pang, B. and L. Lee (2005). Seeing stars: Exploiting class relationships for sentiment
12101 categorization with respect to rating scales. In Proceedings of the Association for
12102 Computational Linguistics (ACL), pp. 115–124.
- 12103 Pang, B. and L. Lee (2008). Opinion mining and sentiment analysis. Foundations and
12104 trends in information retrieval 2(1-2), 1–135.
- 12105 Pang, B., L. Lee, and S. Vaithyanathan (2002). Thumbs up?: sentiment classification using
12106 machine learning techniques. In Proceedings of Empirical Methods for Natural Language
12107 Processing (EMNLP), pp. 79–86.
- 12108 Papineni, K., S. Roukos, T. Ward, and W.-J. Zhu (2002). Bleu: a method for automatic
12109 evaluation of machine translation. In Proceedings of the Association for Computational
12110 Linguistics (ACL), pp. 311–318.
- 12111 Park, J. and C. Cardie (2012). Improving implicit discourse relation recognition through
12112 feature set optimization. In Proceedings of the Special Interest Group on Discourse and
12113 Dialogue (SIGDIAL), pp. 108–112.
- 12114 Parsons, T. (1990). Events in the Semantics of English, Volume 5. MIT Press.

- 12115 Pascanu, R., T. Mikolov, and Y. Bengio (2013). On the difficulty of training recurrent
 12116 neural networks. In Proceedings of the International Conference on Machine Learning
 12117 (ICML), pp. 1310–1318.
- 12118 Paul, M., M. Federico, and S. Stüker (2010). Overview of the iwslt 2010 evaluation
 12119 campaign. In International Workshop on Spoken Language Translation (IWSLT) 2010.
- 12120 Pedersen, T., S. Patwardhan, and J. Michelizzi (2004). Wordnet::similarity - measuring the
 12121 relatedness of concepts. In Proceedings of the North American Chapter of the Association
 12122 for Computational Linguistics (NAACL), pp. 38–41.
- 12123 Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel,
 12124 P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau,
 12125 M. Brucher, M. Perrot, and E. Duchesnay (2011). Scikit-learn: Machine learning in
 12126 Python. Journal of Machine Learning Research 12, 2825–2830.
- 12127 Pei, W., T. Ge, and B. Chang (2015). An effective neural network model for graph-based
 12128 dependency parsing. In Proceedings of the Association for Computational Linguistics
 12129 (ACL), pp. 313–322.
- 12130 Peldszus, A. and M. Stede (2013). From argument diagrams to argumentation mining in
 12131 texts: A survey. International Journal of Cognitive Informatics and Natural Intelligence
 12132 (IJCINI) 7(1), 1–31.
- 12133 Peldszus, A. and M. Stede (2015). An annotated corpus of argumentative microtexts. In
 12134 Proceedings of the First Conference on Argumentation.
- 12135 Peng, F., F. Feng, and A. McCallum (2004). Chinese segmentation and new word detection
 12136 using conditional random fields. In Proceedings of the International Conference on
 12137 Computational Linguistics (COLING), pp. 562.
- 12138 Pennington, J., R. Socher, and C. Manning (2014). Glove: Global vectors for word
 12139 representation. In Proceedings of Empirical Methods for Natural Language Processing
 12140 (EMNLP), pp. 1532–1543.
- 12141 Pereira, F. and Y. Schabes (1992). Inside-outside reestimation from partially bracketed
 12142 corpora. In Proceedings of the Association for Computational Linguistics (ACL), pp.
 12143 128–135.
- 12144 Pereira, F. C. N. and S. M. Shieber (2002). Prolog and natural-language analysis.
 12145 Micromote Publishing.
- 12146 Peters, M. E., M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer
 12147 (2018). Deep contextualized word representations. In Proceedings of the North American
 12148 Chapter of the Association for Computational Linguistics (NAACL).

- 12149 Peterson, W. W., T. G. Birdsall, and W. C. Fox (1954). The theory of signal detectability.
12150 Transactions of the IRE professional group on information theory 4(4), 171–212.
- 12151 Petrov, S., L. Barrett, R. Thibaux, and D. Klein (2006). Learning accurate, compact,
12152 and interpretable tree annotation. In Proceedings of the Association for Computational
12153 Linguistics (ACL).
- 12154 Petrov, S., D. Das, and R. McDonald (2012, May). A universal part-of-speech tagset. In
12155 Proceedings of LREC.
- 12156 Petrov, S. and R. McDonald (2012). Overview of the 2012 shared task on parsing the
12157 web. In Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language
12158 (SANCL), Volume 59.
- 12159 Pinker, S. (2003). *The language instinct: How the mind creates language*. Penguin UK.
- 12160 Pinter, Y., R. Guthrie, and J. Eisenstein (2017). Mimicking word embeddings using
12161 subword RNNs. In Proceedings of Empirical Methods for Natural Language Processing
12162 (EMNLP).
- 12163 Pitler, E., A. Louis, and A. Nenkova (2009). Automatic sense prediction for implicit
12164 discourse relations in text. In Proceedings of the Association for Computational
12165 Linguistics (ACL).
- 12166 Pitler, E. and A. Nenkova (2009). Using syntax to disambiguate explicit discourse
12167 connectives in text. In Proceedings of the Association for Computational Linguistics
12168 (ACL), pp. 13–16.
- 12169 Pitler, E., M. Raghupathy, H. Mehta, A. Nenkova, A. Lee, and A. Joshi (2008). Easily
12170 identifiable discourse relations. In Proceedings of the International Conference on
12171 Computational Linguistics (COLING), pp. 87–90.
- 12172 Plank, B., A. Søgaard, and Y. Goldberg (2016). Multilingual part-of-speech tagging with
12173 bidirectional long short-term memory models and auxiliary loss. In Proceedings of the
12174 Association for Computational Linguistics (ACL).
- 12175 Poesio, M., R. Stevenson, B. Di Eugenio, and J. Hitzeman (2004). Centering: A parametric
12176 theory and its instantiations. *Computational linguistics* 30(3), 309–363.
- 12177 Polanyi, L. and A. Zaenen (2006). Contextual valence shifters. In Computing attitude and
12178 affect in text: Theory and applications. Springer.
- 12179 Ponzetto, S. P. and M. Strube (2006). Exploiting semantic role labeling, wordnet and
12180 wikipedia for coreference resolution. In Proceedings of the North American Chapter of
12181 the Association for Computational Linguistics (NAACL), pp. 192–199.

- 12182 Ponzetto, S. P. and M. Strube (2007). Knowledge derived from wikipedia for computing
 12183 semantic relatedness. *Journal of Artificial Intelligence Research* 30, 181–212.
- 12184 Poon, H. and P. Domingos (2008). Joint unsupervised coreference resolution with markov
 12185 logic. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
 12186 pp. 650–659.
- 12187 Poon, H. and P. Domingos (2009). Unsupervised semantic parsing. In *Proceedings of the
 12188 Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1–10.
- 12189 Popel, M., D. Marecek, J. Stepánek, D. Zeman, and Z. Zabokrtský (2013). Coordination
 12190 structures in dependency treebanks. In *Proceedings of the Association for Computational
 12191 Linguistics (ACL)*, pp. 517–527.
- 12192 Popescu, A.-M., O. Etzioni, and H. Kautz (2003). Towards a theory of natural language
 12193 interfaces to databases. In *Proceedings of Intelligent User Interfaces (IUI)*, pp. 149–157.
- 12194 Poplack, S. (1980). Sometimes i'll start a sentence in spanish y termino en español: toward
 12195 a typology of code-switching1. *Linguistics* 18(7-8), 581–618.
- 12196 Porter, M. F. (1980). An algorithm for suffix stripping. *Program* 14(3), 130–137.
- 12197 Prabhakaran, V., O. Rambow, and M. Diab (2010). Automatic committed belief tagging.
 12198 In *Proceedings of the International Conference on Computational Linguistics (COLING)*,
 12199 pp. 1014–1022.
- 12200 Pradhan, S., X. Luo, M. Recasens, E. Hovy, V. Ng, and M. Strube (2014). Scoring
 12201 coreference partitions of predicted mentions: A reference implementation. In *Proceedings
 12202 of the Association for Computational Linguistics (ACL)*, pp. 30–35.
- 12203 Pradhan, S., L. Ramshaw, M. Marcus, M. Palmer, R. Weischedel, and N. Xue (2011).
 12204 CoNLL-2011 shared task: Modeling unrestricted coreference in OntoNotes. In
 12205 *Proceedings of the Fifteenth Conference on Computational Natural Language Learning:
 12206 Shared Task*, pp. 1–27. Association for Computational Linguistics.
- 12207 Pradhan, S., W. Ward, K. Hacioglu, J. H. Martin, and D. Jurafsky (2005). Semantic
 12208 role labeling using different syntactic views. In *Proceedings of the Association for
 12209 Computational Linguistics (ACL)*, pp. 581–588.
- 12210 Prasad, R., N. Dinesh, A. Lee, E. Miltsakaki, L. Robaldo, A. Joshi, and B. Webber (2008).
 12211 The Penn Discourse Treebank 2.0. In *Proceedings of LREC*.
- 12212 Punyakanok, V., D. Roth, and W.-t. Yih (2008). The importance of syntactic parsing and
 12213 inference in semantic role labeling. *Computational Linguistics* 34(2), 257–287.

- 12214 Pustejovsky, J., P. Hanks, R. Sauri, A. See, R. Gaizauskas, A. Setzer, D. Radev,
12215 B. Sundheim, D. Day, L. Ferro, et al. (2003). The timebank corpus. In *Corpus linguistics*,
12216 Volume 2003, pp. 40. Lancaster, UK.
- 12217 Pustejovsky, J., B. Ingria, R. Sauri, J. Castano, J. Littman, R. Gaizauskas, A. Setzer,
12218 G. Katz, and I. Mani (2005). The specification language timeml. In *The language of*
12219 *time: A reader*, pp. 545–557. Oxford University Press.
- 12220 Qin, L., Z. Zhang, H. Zhao, Z. Hu, and E. Xing (2017). Adversarial connective-exploiting
12221 networks for implicit discourse relation classification. In *Proceedings of the Association*
12222 *for Computational Linguistics (ACL)*, pp. 1006–1017.
- 12223 Qiu, G., B. Liu, J. Bu, and C. Chen (2011). Opinion word expansion and target extraction
12224 through double propagation. *Computational linguistics* 37(1), 9–27.
- 12225 Quattoni, A., S. Wang, L.-P. Morency, M. Collins, and T. Darrell (2007). Hidden
12226 conditional random fields. *IEEE transactions on pattern analysis and machine*
12227 *intelligence* 29(10).
- 12228 Rahman, A. and V. Ng (2011). Narrowing the modeling gap: a cluster-ranking approach
12229 to coreference resolution. *Journal of Artificial Intelligence Research* 40, 469–521.
- 12230 Rajpurkar, P., J. Zhang, K. Lopyrev, and P. Liang (2016). Squad: 100,000+ questions
12231 for machine comprehension of text. In *Proceedings of Empirical Methods for Natural*
12232 *Language Processing (EMNLP)*, pp. 2383–2392.
- 12233 Ranzato, M., S. Chopra, M. Auli, and W. Zaremba (2016). Sequence level training with
12234 recurrent neural networks. In *Proceedings of the International Conference on Learning*
12235 *Representations (ICLR)*.
- 12236 Rao, D., D. Yarowsky, A. Shreevats, and M. Gupta (2010). Classifying latent user attributes
12237 in twitter. In *Proceedings of Workshop on Search and mining user-generated contents*.
- 12238 Ratinov, L. and D. Roth (2009). Design challenges and misconceptions in named entity
12239 recognition. In *Proceedings of the Thirteenth Conference on Computational Natural*
12240 *Language Learning*, pp. 147–155. Association for Computational Linguistics.
- 12241 Ratinov, L., D. Roth, D. Downey, and M. Anderson (2011). Local and global algorithms
12242 for disambiguation to wikipedia. In *Proceedings of the Association for Computational*
12243 *Linguistics (ACL)*, pp. 1375–1384.
- 12244 Ratliff, N. D., J. A. Bagnell, and M. Zinkevich (2007). (approximate) subgradient
12245 methods for structured prediction. In *Proceedings of Artificial Intelligence and Statistics*
12246 *(AISTATS)*, pp. 380–387.

- 12247 Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In emnlp,
12248 pp. 133–142.
- 12249 Ratnaparkhi, A., J. Reynar, and S. Roukos (1994). A maximum entropy model for
12250 prepositional phrase attachment. In Proceedings of the workshop on Human Language
12251 Technology, pp. 250–255. Association for Computational Linguistics.
- 12252 Read, J. (2005). Using emoticons to reduce dependency in machine learning techniques
12253 for sentiment classification. In Proceedings of the ACL student research workshop, pp.
12254 43–48. Association for Computational Linguistics.
- 12255 Reisinger, D., R. Rudinger, F. Ferraro, C. Harman, K. Rawlins, and B. V. Durme (2015).
12256 Semantic proto-roles. *Transactions of the Association for Computational Linguistics* 3,
12257 475–488.
- 12258 Reisinger, J. and R. J. Mooney (2010). Multi-prototype vector-space models of word
12259 meaning. In Proceedings of the North American Chapter of the Association for
12260 Computational Linguistics (NAACL), pp. 109–117.
- 12261 Reiter, E. and R. Dale (2000). Building natural language generation systems. Cambridge
12262 university press.
- 12263 Resnik, P., M. B. Olsen, and M. Diab (1999). The bible as a parallel corpus: Annotating
12264 the ‘book of 2000 tongues’. *Computers and the Humanities* 33(1-2), 129–153.
- 12265 Resnik, P. and N. A. Smith (2003). The web as a parallel corpus. *Computational
12266 Linguistics* 29(3), 349–380.
- 12267 Ribeiro, F. N., M. Araújo, P. Gonçalves, M. A. Gonçalves, and F. Benevenuto (2016).
12268 Sentibench-a benchmark comparison of state-of-the-practice sentiment analysis methods.
12269 EPJ Data Science 5(1), 1–29.
- 12270 Richardson, M., C. J. Burges, and E. Renshaw (2013). MCTest: A challenge dataset for
12271 the open-domain machine comprehension of text. In Proceedings of Empirical Methods
12272 for Natural Language Processing (EMNLP), pp. 193–203.
- 12273 Riedel, S., L. Yao, and A. McCallum (2010). Modeling relations and their mentions without
12274 labeled text. In Proceedings of the European Conference on Machine Learning and
12275 Principles and Practice of Knowledge Discovery in Databases (ECML), pp. 148–163.
- 12276 Riedl, M. O. and R. M. Young (2010). Narrative planning: Balancing plot and character.
12277 *Journal of Artificial Intelligence Research* 39, 217–268.
- 12278 Rieser, V. and O. Lemon (2011). Reinforcement learning for adaptive dialogue systems:
12279 a data-driven methodology for dialogue management and natural language generation.
12280 Springer Science & Business Media.

- 12281 Riloff, E. (1996). Automatically generating extraction patterns from untagged text. In
12282 Proceedings of the national conference on artificial intelligence, pp. 1044–1049.
- 12283 Riloff, E. and J. Wiebe (2003). Learning extraction patterns for subjective expressions. In
12284 Proceedings of the 2003 conference on Empirical methods in natural language processing,
12285 pp. 105–112. Association for Computational Linguistics.
- 12286 Ritchie, G. (2001). Current directions in computational humour. *Artificial Intelligence*
12287 Review 16(2), 119–135.
- 12288 Ritter, A., C. Cherry, and W. B. Dolan (2011). Data-driven response generation in social
12289 media. In Proceedings of Empirical Methods for Natural Language Processing (EMNLP),
12290 pp. 583–593.
- 12291 Ritter, A., S. Clark, Mausam, and O. Etzioni (2011). Named entity recognition in tweets:
12292 an experimental study. In Proceedings of EMNLP.
- 12293 Roark, B., M. Saracclar, and M. Collins (2007). Discriminative n -gram language
12294 modeling. *Computer Speech & Language* 21(2), 373–392.
- 12295 Robert, C. and G. Casella (2013). Monte Carlo statistical methods. Springer Science &
12296 Business Media.
- 12297 Rosenfeld, R. (1996). A maximum entropy approach to adaptive statistical language
12298 modelling. *Computer Speech & Language* 10(3), 187–228.
- 12299 Ross, S., G. Gordon, and D. Bagnell (2011). A reduction of imitation learning and
12300 structured prediction to no-regret online learning. In Proceedings of Artificial Intelligence
12301 and Statistics (AISTATS), pp. 627–635.
- 12302 Roy, N., J. Pineau, and S. Thrun (2000). Spoken dialogue management using probabilistic
12303 reasoning. In Proceedings of the Association for Computational Linguistics (ACL), pp.
12304 93–100.
- 12305 Rudnicky, A. and W. Xu (1999). An agenda-based dialog management architecture for
12306 spoken language systems. In IEEE Automatic Speech Recognition and Understanding
12307 Workshop, Volume 13.
- 12308 Rush, A. M., S. Chopra, and J. Weston (2015). A neural attention model for abstractive
12309 sentence summarization. In Proceedings of Empirical Methods for Natural Language
12310 Processing (EMNLP), pp. 379–389.
- 12311 Rush, A. M., D. Sontag, M. Collins, and T. Jaakkola (2010). On dual decomposition
12312 and linear programming relaxations for natural language processing. In Proceedings of
12313 Empirical Methods for Natural Language Processing (EMNLP), pp. 1–11.

- 12314 Russell, S. J. and P. Norvig (2009). Artificial intelligence: a modern approach (3rd ed.).
 12315 Prentice Hall.
- 12316 Rutherford, A., V. Demberg, and N. Xue (2017). A systematic study of neural discourse
 12317 models for implicit discourse relation. In Proceedings of the European Chapter of the
 12318 Association for Computational Linguistics (EACL), pp. 281–291.
- 12319 Rutherford, A. T. and N. Xue (2014). Discovering implicit discourse relations through
 12320 brown cluster pair representation and coreference patterns. In Proceedings of the
 12321 European Chapter of the Association for Computational Linguistics (EACL).
- 12322 Sag, I. A., T. Baldwin, F. Bond, A. Copestake, and D. Flickinger (2002). Multiword
 12323 expressions: A pain in the neck for nlp. In International Conference on Intelligent Text
 12324 Processing and Computational Linguistics, pp. 1–15. Springer.
- 12325 Sagae, K. (2009). Analysis of discourse structure with syntactic dependencies and
 12326 data-driven shift-reduce parsing. In Proceedings of the 11th International Conference on
 12327 Parsing Technologies, pp. 81–84.
- 12328 Santos, C. D. and B. Zadrozny (2014). Learning character-level representations for
 12329 part-of-speech tagging. In Proceedings of the International Conference on Machine
 12330 Learning (ICML), pp. 1818–1826.
- 12331 Sato, M.-A. and S. Ishii (2000). On-line em algorithm for the normalized gaussian network.
 12332 Neural computation 12(2), 407–432.
- 12333 Saurí, R. and J. Pustejovsky (2009). Factbank: a corpus annotated with event factuality.
 12334 Language resources and evaluation 43(3), 227.
- 12335 Saxe, A. M., J. L. McClelland, and S. Ganguli (2014). Exact solutions to the nonlinear
 12336 dynamics of learning in deep linear neural networks. In Proceedings of the International
 12337 Conference on Learning Representations (ICLR).
- 12338 Schank, R. C. and R. Abelson (1977). Scripts, goals, plans, and understanding. Hillsdale,
 12339 NJ: Erlbaum.
- 12340 Schapire, R. E. and Y. Singer (2000). Boostexter: A boosting-based system for text
 12341 categorization. Machine learning 39(2-3), 135–168.
- 12342 Schaul, T., S. Zhang, and Y. LeCun (2013). No more pesky learning rates. In Proceedings
 12343 of the International Conference on Machine Learning (ICML), pp. 343–351.
- 12344 Schnabel, T., I. Labutov, D. Mimno, and T. Joachims (2015). Evaluation methods for
 12345 unsupervised word embeddings. In Proceedings of Empirical Methods for Natural
 12346 Language Processing (EMNLP), pp. 298–307.

- 12347 Schneider, N., J. Flanigan, and T. O'Gorman (2015). The logic of amr: Practical, unified,
12348 graph-based sentence semantics for nlp. In Proceedings of the North American Chapter
12349 of the Association for Computational Linguistics (NAACL), pp. 4–5.
- 12350 Schütze, H. (1998). Automatic word sense discrimination. *Computational linguistics* 24(1),
12351 97–123.
- 12352 Schwarm, S. E. and M. Ostendorf (2005). Reading level assessment using support
12353 vector machines and statistical language models. In Proceedings of the Association
12354 for Computational Linguistics (ACL), pp. 523–530.
- 12355 See, A., P. J. Liu, and C. D. Manning (2017). Get to the point: Summarization
12356 with pointer-generator networks. In Proceedings of the Association for Computational
12357 Linguistics (ACL), pp. 1073–1083.
- 12358 Sennrich, R., B. Haddow, and A. Birch (2016). Neural machine translation of rare words
12359 with subword units. In Proceedings of the Association for Computational Linguistics
12360 (ACL), pp. 1715–1725.
- 12361 Serban, I. V., A. Sordoni, Y. Bengio, A. C. Courville, and J. Pineau (2016). Building
12362 end-to-end dialogue systems using generative hierarchical neural network models. In
12363 Proceedings of the National Conference on Artificial Intelligence (AAAI), pp. 3776–3784.
- 12364 Settles, B. (2012). Active learning. *Synthesis Lectures on Artificial Intelligence and Machine*
12365 *Learning* 6(1), 1–114.
- 12366 Shang, L., Z. Lu, and H. Li (2015). Neural responding machine for short-text conversation.
12367 In Proceedings of the Association for Computational Linguistics (ACL), pp. 1577–1586.
- 12368 Shen, D. and M. Lapata (2007). Using semantic roles to improve question answering.
12369 In Proceedings of Empirical Methods for Natural Language Processing (EMNLP), pp.
12370 12–21.
- 12371 Shen, S., Y. Cheng, Z. He, W. He, H. Wu, M. Sun, and Y. Liu (2016). Minimum
12372 risk training for neural machine translation. In Proceedings of the Association for
12373 Computational Linguistics (ACL), pp. 1683–1692.
- 12374 Shen, W., J. Wang, and J. Han (2015). Entity linking with a knowledge base: Issues,
12375 techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering* 27(2),
12376 443–460.
- 12377 Schieber, S. M. (1985). Evidence against the context-freeness of natural language.
12378 *Linguistics and Philosophy* 8(3), 333–343.

- 12379 Siegelmann, H. T. and E. D. Sontag (1995). On the computational power of neural nets.
 12380 Journal of computer and system sciences 50(1), 132–150.
- 12381 Singh, S., A. Subramanya, F. Pereira, and A. McCallum (2011). Large-scale cross-document
 12382 coreference using distributed inference and hierarchical models. In Proceedings of the
 12383 Association for Computational Linguistics (ACL), pp. 793–803.
- 12384 Sipser, M. (2012). Introduction to the Theory of Computation. Cengage Learning.
- 12385 Smith, D. A. and J. Eisner (2006). Minimum risk annealing for training log-linear models.
 12386 In Proceedings of the Association for Computational Linguistics (ACL), pp. 787–794.
- 12387 Smith, D. A. and J. Eisner (2008). Dependency parsing by belief propagation. In
 12388 Proceedings of Empirical Methods for Natural Language Processing (EMNLP), pp. 145–
 12389 156.
- 12390 Smith, D. A. and N. A. Smith (2007). Probabilistic models of nonprojective dependency
 12391 trees. In Proceedings of Empirical Methods for Natural Language Processing (EMNLP),
 12392 pp. 132–140.
- 12393 Smith, N. A. (2011). Linguistic structure prediction. Synthesis Lectures on Human
 12394 Language Technologies 4(2), 1–274.
- 12395 Snover, M., B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul (2006). A study of
 12396 translation edit rate with targeted human annotation. In Proceedings of association
 12397 for machine translation in the Americas, Volume 200.
- 12398 Snow, R., B. O’Connor, D. Jurafsky, and A. Y. Ng (2008). Cheap and fast—but is it
 12399 good?: evaluating non-expert annotations for natural language tasks. In Proceedings of
 12400 Empirical Methods for Natural Language Processing (EMNLP), pp. 254–263.
- 12401 Snyder, B. and R. Barzilay (2007). Database-text alignment via structured multilabel
 12402 classification. In Proceedings of the International Joint Conference on Artificial
 12403 Intelligence (IJCAI), pp. 1713–1718.
- 12404 Socher, R., J. Bauer, C. D. Manning, and A. Y. Ng (2013). Parsing with compositional
 12405 vector grammars. In Proceedings of the Association for Computational Linguistics
 12406 (ACL).
- 12407 Socher, R., B. Huval, C. D. Manning, and A. Y. Ng (2012). Semantic compositionality
 12408 through recursive matrix-vector spaces. In Proceedings of the 2012 Joint Conference
 12409 on Empirical Methods in Natural Language Processing and Computational Natural
 12410 Language Learning, pp. 1201–1211. Association for Computational Linguistics.

- 12411 Socher, R., A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts
12412 (2013). Recursive deep models for semantic compositionality over a sentiment treebank.
12413 In Proceedings of Empirical Methods for Natural Language Processing (EMNLP).
- 12414 Søgaard, A. (2013). Semi-supervised learning and domain adaptation in natural language
12415 processing. *Synthesis Lectures on Human Language Technologies* 6(2), 1–103.
- 12416 Solorio, T. and Y. Liu (2008). Learning to predict code-switching points. In Proceedings of
12417 Empirical Methods for Natural Language Processing (EMNLP), pp. 973–981. Association
12418 for Computational Linguistics.
- 12419 Somasundaran, S., G. Namata, J. Wiebe, and L. Getoor (2009). Supervised and
12420 unsupervised methods in employing discourse relations for improving opinion polarity
12421 classification. In Proceedings of Empirical Methods for Natural Language Processing
12422 (EMNLP).
- 12423 Somasundaran, S. and J. Wiebe (2009). Recognizing stances in online debates. In
12424 Proceedings of the Association for Computational Linguistics (ACL), pp. 226–234.
- 12425 Song, L., B. Boots, S. M. Siddiqi, G. J. Gordon, and A. J. Smola (2010). Hilbert space
12426 embeddings of hidden markov models. In Proceedings of the International Conference
12427 on Machine Learning (ICML), pp. 991–998.
- 12428 Song, L., Y. Zhang, X. Peng, Z. Wang, and D. Gildea (2016). Amr-to-text generation as a
12429 traveling salesman problem. In Proceedings of Empirical Methods for Natural Language
12430 Processing (EMNLP), pp. 2084–2089.
- 12431 Soon, W. M., H. T. Ng, and D. C. Y. Lim (2001). A machine learning approach to
12432 coreference resolution of noun phrases. *Computational linguistics* 27(4), 521–544.
- 12433 Sordoni, A., M. Galley, M. Auli, C. Brockett, Y. Ji, M. Mitchell, J.-Y. Nie, J. Gao,
12434 and B. Dolan (2015). A neural network approach to context-sensitive generation
12435 of conversational responses. In Proceedings of the North American Chapter of the
12436 Association for Computational Linguistics (NAACL).
- 12437 Soricut, R. and D. Marcu (2003). Sentence level discourse parsing using syntactic and
12438 lexical information. In Proceedings of the North American Chapter of the Association
12439 for Computational Linguistics (NAACL), pp. 149–156.
- 12440 Sowa, J. F. (2000). Knowledge representation: logical, philosophical, and computational
12441 foundations. Pacific Grove, CA: Brooks/Cole.
- 12442 Spärck Jones, K. (1972). A statistical interpretation of term specificity and its application
12443 in retrieval. *Journal of documentation* 28(1), 11–21.

- 12444 Spitkovsky, V. I., H. Alshawi, D. Jurafsky, and C. D. Manning (2010). Viterbi training
 12445 improves unsupervised dependency parsing. In CONLL, pp. 9–17.
- 12446 Sporleder, C. and M. Lapata (2005). Discourse chunking and its application to sentence
 12447 compression. In Proceedings of Empirical Methods for Natural Language Processing
 12448 (EMNLP), pp. 257–264.
- 12449 Sproat, R., A. Black, S. Chen, S. Kumar, M. Ostendorf, and C. Richards (2001).
 12450 Normalization of non-standard words. Computer Speech & Language 15(3), 287–333.
- 12451 Sproat, R., W. Gale, C. Shih, and N. Chang (1996). A stochastic finite-state
 12452 word-segmentation algorithm for chinese. Computational linguistics 22(3), 377–404.
- 12453 Sra, S., S. Nowozin, and S. J. Wright (2012). Optimization for machine learning. MIT
 12454 Press.
- 12455 Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014).
 12456 Dropout: A simple way to prevent neural networks from overfitting. The Journal of
 12457 Machine Learning Research 15(1), 1929–1958.
- 12458 Srivastava, R. K., K. Greff, and J. Schmidhuber (2015). Training very deep networks. In
 12459 Neural Information Processing Systems (NIPS), pp. 2377–2385.
- 12460 Stab, C. and I. Gurevych (2014a). Annotating argument components and relations in
 12461 persuasive essays. In Proceedings of the International Conference on Computational
 12462 Linguistics (COLING), pp. 1501–1510.
- 12463 Stab, C. and I. Gurevych (2014b). Identifying argumentative discourse structures in
 12464 persuasive essays. In Proceedings of the 2014 Conference on Empirical Methods in
 12465 Natural Language Processing (EMNLP), pp. 46–56.
- 12466 Stede, M. (2011, nov). Discourse Processing, Volume 4 of Synthesis Lectures on Human
 12467 Language Technologies. Morgan & Claypool Publishers.
- 12468 Steedman, M. and J. Baldridge (2011). Combinatory categorial grammar.
 12469 In Non-Transformational Syntax: Formal and Explicit Models of Grammar.
 12470 Wiley-Blackwell.
- 12471 Stenetorp, P., S. Pyysalo, G. Topić, T. Ohta, S. Ananiadou, and J. Tsujii (2012). Brat: a
 12472 web-based tool for nlp-assisted text annotation. In Proceedings of the European Chapter
 12473 of the Association for Computational Linguistics (EACL), pp. 102–107.
- 12474 Stern, M., J. Andreas, and D. Klein (2017). A minimal span-based neural constituency
 12475 parser. In Proceedings of the Association for Computational Linguistics (ACL).

- 12476 Stolcke, A., K. Ries, N. Coccaro, E. Shriberg, R. Bates, D. Jurafsky, P. Taylor, R. Martin,
12477 C. Van Ess-Dykema, and M. Meteer (2000). Dialogue act modeling for automatic tagging
12478 and recognition of conversational speech. *Computational linguistics* 26(3), 339–373.
- 12479 Stone, P. J. (1966). *The General Inquirer: A Computer Approach to Content Analysis*.
12480 The MIT Press.
- 12481 Stoyanov, V., N. Gilbert, C. Cardie, and E. Riloff (2009). Conundrums in noun phrase
12482 coreference resolution: Making sense of the state-of-the-art. In Proceedings of the
12483 Association for Computational Linguistics (ACL), pp. 656–664.
- 12484 Strang, G. (2016). *Introduction to linear algebra* (Fifth ed.). Wellesley, MA:
12485 Wellesley-Cambridge Press.
- 12486 Strubell, E., P. Verga, D. Belanger, and A. McCallum (2017). Fast and accurate entity
12487 recognition with iterated dilated convolutions. In Proceedings of Empirical Methods for
12488 Natural Language Processing (EMNLP).
- 12489 Suchanek, F. M., G. Kasneci, and G. Weikum (2007). Yago: a core of semantic knowledge.
12490 In Proceedings of the Conference on World-Wide Web (WWW), pp. 697–706.
- 12491 Sun, X., T. Matsuzaki, D. Okanohara, and J. Tsujii (2009). Latent variable perceptron
12492 algorithm for structured classification. In Proceedings of the International Joint
12493 Conference on Artificial Intelligence (IJCAI), Volume 9, pp. 1236–1242.
- 12494 Sun, Y., L. Lin, D. Tang, N. Yang, Z. Ji, and X. Wang (2015). Modeling mention, context
12495 and entity with neural networks for entity disambiguation. In IJCAI, pp. 1333–1339.
- 12496 Sundermeyer, M., R. Schlüter, and H. Ney (2012). Lstm neural networks for language
12497 modeling. In INTERSPEECH.
- 12498 Surdeanu, M., J. Tibshirani, R. Nallapati, and C. D. Manning (2012). Multi-instance
12499 multi-label learning for relation extraction. In Proceedings of Empirical Methods for
12500 Natural Language Processing (EMNLP), pp. 455–465.
- 12501 Sutskever, I., O. Vinyals, and Q. V. Le (2014). Sequence to sequence learning with neural
12502 networks. In Neural Information Processing Systems (NIPS), pp. 3104–3112.
- 12503 Sutton, R. S. and A. G. Barto (1998). Reinforcement learning: An introduction, Volume 1.
12504 MIT press Cambridge.
- 12505 Sutton, R. S., D. A. McAllester, S. P. Singh, and Y. Mansour (2000). Policy gradient
12506 methods for reinforcement learning with function approximation. In Neural Information
12507 Processing Systems (NIPS), pp. 1057–1063.

- 12508 Taboada, M., J. Brooke, M. Tofiloski, K. Voll, and M. Stede (2011). Lexicon-based methods
 12509 for sentiment analysis. *Computational linguistics* 37(2), 267–307.
- 12510 Taboada, M. and W. C. Mann (2006). Rhetorical structure theory: Looking back and
 12511 moving ahead. *Discourse studies* 8(3), 423–459.
- 12512 Täckström, O., K. Ganchev, and D. Das (2015). Efficient inference and structured
 12513 learning for semantic role labeling. *Transactions of the Association for Computational
 12514 Linguistics* 3, 29–41.
- 12515 Täckström, O., R. McDonald, and J. Uszkoreit (2012). Cross-lingual word clusters for
 12516 direct transfer of linguistic structure. In *Proceedings of the North American Chapter of
 12517 the Association for Computational Linguistics (NAACL)*, pp. 477–487.
- 12518 Tang, D., B. Qin, and T. Liu (2015). Document modeling with gated recurrent neural
 12519 network for sentiment classification. In *Proceedings of Empirical Methods for Natural
 12520 Language Processing (EMNLP)*, pp. 1422–1432.
- 12521 Taskar, B., C. Guestrin, and D. Koller (2003). Max-margin markov networks. In *Neural
 12522 Information Processing Systems (NIPS)*.
- 12523 Tausczik, Y. R. and J. W. Pennebaker (2010). The psychological meaning of words:
 12524 LIWC and computerized text analysis methods. *Journal of Language and Social
 12525 Psychology* 29(1), 24–54.
- 12526 Teh, Y. W. (2006). A hierarchical bayesian language model based on pitman-yor processes.
 12527 In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 985–992.
- 12528 Tesnière, L. (1966). *Éléments de syntaxe structurale* (second ed.). Paris: Klincksieck.
- 12529 Teufel, S., J. Carletta, and M. Moens (1999). An annotation scheme for discourse-level
 12530 argumentation in research articles. In *Proceedings of the European Chapter of the
 12531 Association for Computational Linguistics (EACL)*, pp. 110–117.
- 12532 Teufel, S. and M. Moens (2002). Summarizing scientific articles: experiments with relevance
 12533 and rhetorical status. *Computational linguistics* 28(4), 409–445.
- 12534 Thomas, M., B. Pang, and L. Lee (2006). Get out the vote: Determining support or
 12535 opposition from Congressional floor-debate transcripts. In *Proceedings of Empirical
 12536 Methods for Natural Language Processing (EMNLP)*, pp. 327–335.
- 12537 Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the
 12538 Royal Statistical Society. Series B (Methodological)*, 267–288.

- 12539 Titov, I. and J. Henderson (2007). Constituent parsing with incremental sigmoid belief
12540 networks. In Proceedings of the Association for Computational Linguistics (ACL), pp.
12541 632–639.
- 12542 Toutanova, K., D. Klein, C. D. Manning, and Y. Singer (2003). Feature-rich part-of-speech
12543 tagging with a cyclic dependency network. In Proceedings of the North American
12544 Chapter of the Association for Computational Linguistics (NAACL).
- 12545 Trivedi, R. and J. Eisenstein (2013). Discourse connectors for latent subjectivity in
12546 sentiment analysis. In Proceedings of the North American Chapter of the Association
12547 for Computational Linguistics (NAACL), pp. 808–813.
- 12548 Tromble, R. W. and J. Eisner (2006). A fast finite-state relaxation method for enforcing
12549 global constraints on sequence decoding. In Proceedings of the North American Chapter
12550 of the Association for Computational Linguistics (NAACL), pp. 423.
- 12551 Tschantaridis, I., T. Hofmann, T. Joachims, and Y. Altun (2004). Support vector
12552 machine learning for interdependent and structured output spaces. In Proceedings of
12553 the twenty-first international conference on Machine learning, pp. 104. ACM.
- 12554 Tsvetkov, Y., M. Faruqui, W. Ling, G. Lample, and C. Dyer (2015). Evaluation of word
12555 vector representations by subspace alignment. In Proceedings of Empirical Methods for
12556 Natural Language Processing (EMNLP), pp. 2049–2054.
- 12557 Tu, Z., Z. Lu, Y. Liu, X. Liu, and H. Li (2016). Modeling coverage for neural machine
12558 translation. In Proceedings of the Association for Computational Linguistics (ACL), pp.
12559 76–85.
- 12560 Turian, J., L. Ratinov, and Y. Bengio (2010). Word representations: a simple and
12561 general method for semi-supervised learning. In Proceedings of the Association for
12562 Computational Linguistics (ACL), pp. 384–394.
- 12563 Turing, A. M. (2009). Computing machinery and intelligence. In Parsing the Turing Test,
12564 pp. 23–65. Springer.
- 12565 Turney, P. D. and P. Pantel (2010). From frequency to meaning: Vector space models of
12566 semantics. Journal of Artificial Intelligence Research 37, 141–188.
- 12567 Tutin, A. and R. Kittredge (1992). Lexical choice in context: generating procedural texts.
12568 In Proceedings of the International Conference on Computational Linguistics (COLING),
12569 pp. 763–769.
- 12570 Twain, M. (1997). A Tramp Abroad. New York: Penguin.

- 12571 Tzeng, E., J. Hoffman, T. Darrell, and K. Saenko (2015). Simultaneous deep transfer across
12572 domains and tasks. In Proceedings of the IEEE International Conference on Computer
12573 Vision, pp. 4068–4076.
- 12574 Usunier, N., D. Buffoni, and P. Gallinari (2009). Ranking with ordered weighted pairwise
12575 classification. In Proceedings of the International Conference on Machine Learning
12576 (ICML), pp. 1057–1064.
- 12577 Uthus, D. C. and D. W. Aha (2013). The ubuntu chat corpus for multiparicipant chat
12578 analysis. In AAAI Spring Symposium: Analyzing Microtext, Volume 13, pp. 01.
- 12579 Utiyama, M. and H. Isahara (2001). A statistical model for domain-independent
12580 text segmentation. In Proceedings of the 39th Annual Meeting on Association for
12581 Computational Linguistics, pp. 499–506. Association for Computational Linguistics.
- 12582 Utiyama, M. and H. Isahara (2007). A comparison of pivot methods for phrase-based
12583 statistical machine translation. In Human Language Technologies 2007: The Conference
12584 of the North American Chapter of the Association for Computational Linguistics;
12585 Proceedings of the Main Conference, pp. 484–491.
- 12586 Uzuner, Ö., X. Zhang, and T. Sibanda (2009). Machine learning and rule-based
12587 approaches to assertion classification. Journal of the American Medical Informatics
12588 Association 16(1), 109–115.
- 12589 Vadas, D. and J. R. Curran (2011). Parsing noun phrases in the penn treebank.
12590 Computational Linguistics 37(4), 753–809.
- 12591 Van Eynde, F. (2006). NP-internal agreement and the structure of the noun phrase. Journal
12592 of Linguistics 42(1), 139–186.
- 12593 Van Gael, J., A. Vlachos, and Z. Ghahramani (2009). The infinite hmm for unsupervised
12594 pos tagging. In Proceedings of Empirical Methods for Natural Language Processing
12595 (EMNLP), pp. 678–687.
- 12596 Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser,
12597 and I. Polosukhin (2017). Attention is all you need. In Neural Information Processing
12598 Systems (NIPS), pp. 6000–6010.
- 12599 Velldal, E., L. Øvrelid, J. Read, and S. Oepen (2012). Speculation and negation: Rules,
12600 rankers, and the role of syntax. Computational linguistics 38(2), 369–410.
- 12601 Versley, Y. (2011). Towards finer-grained tagging of discourse connectives. In Proceedings
12602 of the Workshop Beyond Semantics: Corpus-based Investigations of Pragmatic and
12603 Discourse Phenomena, pp. 2–63.

- 12604 Vilain, M., J. Burger, J. Aberdeen, D. Connolly, and L. Hirschman (1995). A
12605 model-theoretic coreference scoring scheme. In Proceedings of the 6th conference on
12606 Message understanding, pp. 45–52. Association for Computational Linguistics.
- 12607 Vincent, P., H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol (2010). Stacked
12608 denoising autoencoders: Learning useful representations in a deep network with a local
12609 denoising criterion. *Journal of Machine Learning Research* 11(Dec), 3371–3408.
- 12610 Vincze, V., G. Szarvas, R. Farkas, G. Móra, and J. Csirik (2008). The bioscope
12611 corpus: biomedical texts annotated for uncertainty, negation and their scopes. *BMC*
12612 *bioinformatics* 9(11), S9.
- 12613 Vinyals, O., A. Toshev, S. Bengio, and D. Erhan (2015). Show and tell: A neural image
12614 caption generator. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE*
12615 Conference on, pp. 3156–3164. IEEE.
- 12616 Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum
12617 decoding algorithm. *IEEE transactions on Information Theory* 13(2), 260–269.
- 12618 Voll, K. and M. Taboada (2007). Not all words are created equal: Extracting semantic
12619 orientation as a function of adjective relevance. In *Proceedings of Australian Conference*
12620 *on Artificial Intelligence*.
- 12621 Wager, S., S. Wang, and P. S. Liang (2013). Dropout training as adaptive regularization.
12622 In *Neural Information Processing Systems (NIPS)*, pp. 351–359.
- 12623 Wainwright, M. J. and M. I. Jordan (2008). Graphical models, exponential families, and
12624 variational inference. *Foundations and Trends® in Machine Learning* 1(1-2), 1–305.
- 12625 Walker, M. A. (2000). An application of reinforcement learning to dialogue strategy
12626 selection in a spoken dialogue system for email. *Journal of Artificial Intelligence*
12627 *Research* 12, 387–416.
- 12628 Walker, M. A., J. E. Cahn, and S. J. Whittaker (1997). Improvising linguistic style:
12629 Social and affective bases for agent personality. In *Proceedings of the first international*
12630 *conference on Autonomous agents*, pp. 96–105. ACM.
- 12631 Wang, C., N. Xue, and S. Pradhan (2015). A Transition-based Algorithm for AMR Parsing.
12632 In *Proceedings of the North American Chapter of the Association for Computational*
12633 *Linguistics (NAACL)*, pp. 366–375.
- 12634 Wang, H., T. Onishi, K. Gimpel, and D. McAllester (2017). Emergent predication structure
12635 in hidden state vectors of neural readers. In *Proceedings of the 2nd Workshop on*
12636 *Representation Learning for NLP*, pp. 26–36.

- 12637 Weaver, W. (1955). Translation. *Machine translation of languages* 14, 15–23.
- 12638 Webber, B. (2004, sep). D-LTAG: extending lexicalized TAG to discourse. *Cognitive Science* 28(5), 751–779.
- 12640 Webber, B., M. Egg, and V. Kordoni (2012). Discourse structure and language technology. *Journal of Natural Language Engineering* 1.
- 12642 Webber, B. and A. Joshi (2012). Discourse structure and computation: past, present and future. In *Proceedings of the ACL-2012 Special Workshop on Rediscovering 50 Years of Discoveries*, pp. 42–54. Association for Computational Linguistics.
- 12645 Wei, G. C. and M. A. Tanner (1990). A monte carlo implementation of the em algorithm and the poor man’s data augmentation algorithms. *Journal of the American Statistical Association* 85(411), 699–704.
- 12648 Weinberger, K., A. Dasgupta, J. Langford, A. Smola, and J. Attenberg (2009). Feature hashing for large scale multitask learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1113–1120.
- 12651 Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM* 9(1), 36–45.
- 12653 Wellner, B. and J. Pustejovsky (2007). Automatically identifying the arguments of discourse connectives. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 92–101.
- 12656 Wen, T.-H., M. Gasic, N. Mrkšić, P.-H. Su, D. Vandyke, and S. Young (2015). Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1711–1721.
- 12660 Weston, J., S. Bengio, and N. Usunier (2011). Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, Volume 11, pp. 2764–2770.
- 12662 Wiebe, J., T. Wilson, and C. Cardie (2005). Annotating expressions of opinions and emotions in language. *Language resources and evaluation* 39(2), 165–210.
- 12664 Wieting, J., M. Bansal, K. Gimpel, and K. Livescu (2015). Towards universal paraphrastic sentence embeddings. *arXiv preprint arXiv:1511.08198*.
- 12666 Wieting, J., M. Bansal, K. Gimpel, and K. Livescu (2016). CHARAGRAM: Embedding words and sentences via character n-grams. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1504–1515.

- 12669 Williams, J. D. and S. Young (2007). Partially observable markov decision processes for
12670 spoken dialog systems. *Computer Speech & Language* 21(2), 393–422.
- 12671 Williams, P., R. Sennrich, M. Post, and P. Koehn (2016). Syntax-based statistical machine
12672 translation. *Synthesis Lectures on Human Language Technologies* 9(4), 1–208.
- 12673 Wilson, T., J. Wiebe, and P. Hoffmann (2005). Recognizing contextual polarity in
12674 phrase-level sentiment analysis. In *Proceedings of Empirical Methods for Natural
12675 Language Processing (EMNLP)*, pp. 347–354.
- 12676 Winograd, T. (1972). Understanding natural language. *Cognitive psychology* 3(1), 1–191.
- 12677 Wiseman, S., A. M. Rush, and S. M. Shieber (2016). Learning global features for coreference
12678 resolution. In *Proceedings of the North American Chapter of the Association for
12679 Computational Linguistics (NAACL)*, pp. 994–1004.
- 12680 Wiseman, S., S. Shieber, and A. Rush (2017). Challenges in data-to-document generation.
12681 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.
12682 2253–2263.
- 12683 Wiseman, S. J., A. M. Rush, S. M. Shieber, and J. Weston (2015). Learning anaphoricity
12684 and antecedent ranking features for coreference resolution. In *Proceedings of the
12685 Association for Computational Linguistics (ACL)*.
- 12686 Wolf, F. and E. Gibson (2005). Representing discourse coherence: A corpus-based study.
12687 *Computational Linguistics* 31(2), 249–287.
- 12688 Wolfe, T., M. Dredze, and B. Van Durme (2017). Pocket knowledge base population. In
12689 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 305–310.
- 12690 Wong, Y. W. and R. Mooney (2007). Generation by inverting a semantic parser that uses
12691 statistical machine translation. In *Proceedings of the North American Chapter of the
12692 Association for Computational Linguistics (NAACL)*, pp. 172–179.
- 12693 Wong, Y. W. and R. J. Mooney (2006). Learning for semantic parsing with statistical
12694 machine translation. In *Proceedings of the North American Chapter of the Association
12695 for Computational Linguistics (NAACL)*, pp. 439–446.
- 12696 Wu, B. Y. and K.-M. Chao (2004). Spanning trees and optimization problems. CRC Press.
- 12697 Wu, D. (1997). Stochastic inversion transduction grammars and bilingual parsing of parallel
12698 corpora. *Computational linguistics* 23(3), 377–403.
- 12699 Wu, F. and D. S. Weld (2010). Open information extraction using wikipedia. In *Proceedings
12700 of the Association for Computational Linguistics (ACL)*, pp. 118–127.

- 12701 Wu, X., R. Ward, and L. Bottou (2018). Wngrad: Learn the learning rate in gradient
 12702 descent. arXiv preprint arXiv:1803.02865.
- 12703 Wu, Y., M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao,
 12704 Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser,
 12705 S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang,
 12706 C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and
 12707 J. Dean (2016). Google’s neural machine translation system: Bridging the gap between
 12708 human and machine translation. CoRR abs/1609.08144.
- 12709 Xia, F. (2000). The part-of-speech tagging guidelines for the penn chinese treebank (3.0).
 12710 Technical report, University of Pennsylvania Institute for Research in Cognitive Science.
- 12711 Xu, K., J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio
 12712 (2015). Show, attend and tell: Neural image caption generation with visual attention.
 12713 In Proceedings of the International Conference on Machine Learning (ICML), pp. 2048–
 12714 2057.
- 12715 Xu, W., X. Liu, and Y. Gong (2003). Document clustering based on non-negative matrix
 12716 factorization. In SIGIR, pp. 267–273. ACM.
- 12717 Xu, Y., L. Mou, G. Li, Y. Chen, H. Peng, and Z. Jin (2015). Classifying relations via
 12718 long short term memory networks along shortest dependency paths. In Proceedings of
 12719 Empirical Methods for Natural Language Processing (EMNLP), pp. 1785–1794.
- 12720 Xuan Bach, N., N. L. Minh, and A. Shimazu (2012). A reranking model for discourse
 12721 segmentation using subtree features. In Proceedings of the Special Interest Group on
 12722 Discourse and Dialogue (SIGDIAL).
- 12723 Xue, N. et al. (2003). Chinese word segmentation as character tagging. Computational
 12724 Linguistics and Chinese Language Processing 8(1), 29–48.
- 12725 Xue, N., H. T. Ng, S. Pradhan, R. Prasad, C. Bryant, and A. T. Rutherford (2015). The
 12726 CoNLL-2015 shared task on shallow discourse parsing. In Proceedings of the Conference
 12727 on Natural Language Learning (CoNLL).
- 12728 Xue, N., H. T. Ng, S. Pradhan, A. Rutherford, B. L. Webber, C. Wang, and H. Wang
 12729 (2016). Conll 2016 shared task on multilingual shallow discourse parsing. In CoNLL
 12730 Shared Task, pp. 1–19.
- 12731 Yamada, H. and Y. Matsumoto (2003). Statistical dependency analysis with support vector
 12732 machines. In Proceedings of IWPT, Volume 3, pp. 195–206.

- 12733 Yamada, K. and K. Knight (2001). A syntax-based statistical translation model. In
12734 Proceedings of the 39th Annual Meeting on Association for Computational Linguistics,
12735 pp. 523–530. Association for Computational Linguistics.
- 12736 Yang, B. and C. Cardie (2014). Context-aware learning for sentence-level sentiment analysis
12737 with posterior regularization. In Proceedings of the Association for Computational
12738 Linguistics (ACL).
- 12739 Yang, Y., M.-W. Chang, and J. Eisenstein (2016). Toward socially-infused information
12740 extraction: Embedding authors, mentions, and entities. In Proceedings of Empirical
12741 Methods for Natural Language Processing (EMNLP).
- 12742 Yang, Y. and J. Eisenstein (2013). A log-linear model for unsupervised text normalization.
12743 In Proceedings of Empirical Methods for Natural Language Processing (EMNLP).
- 12744 Yang, Y. and J. Eisenstein (2015). Unsupervised multi-domain adaptation with feature
12745 embeddings. In Proceedings of the North American Chapter of the Association for
12746 Computational Linguistics (NAACL).
- 12747 Yang, Y., W.-t. Yih, and C. Meek (2015). WikiQA: A challenge dataset for open-domain
12748 question answering. In Proceedings of Empirical Methods for Natural Language
12749 Processing (EMNLP), pp. 2013–2018.
- 12750 Yannakoudakis, H., T. Briscoe, and B. Medlock (2011). A new dataset and method for
12751 automatically grading esol texts. In Proceedings of the 49th Annual Meeting of the
12752 Association for Computational Linguistics: Human Language Technologies-Volume 1,
12753 pp. 180–189. Association for Computational Linguistics.
- 12754 Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods.
12755 In Proceedings of the Association for Computational Linguistics (ACL), pp. 189–196.
12756 Association for Computational Linguistics.
- 12757 Yee, L. C. and T. Y. Jones (2012, March). Apple ceo in china mission to clear up problems.
12758 Reuters. retrieved on March 26, 2017.
- 12759 Yi, Y., C.-Y. Lai, S. Petrov, and K. Keutzer (2011, October). Efficient parallel cky parsing
12760 on gpus. In Proceedings of the 12th International Conference on Parsing Technologies,
12761 Dublin, Ireland, pp. 175–185. Association for Computational Linguistics.
- 12762 Yu, C.-N. J. and T. Joachims (2009). Learning structural svms with latent variables. In
12763 Proceedings of the International Conference on Machine Learning (ICML), pp. 1169–
12764 1176.
- 12765 Yu, F. and V. Koltun (2016). Multi-scale context aggregation by dilated convolutions. In
12766 Proceedings of the International Conference on Learning Representations (ICLR).

- 12767 Zaidan, O. F. and C. Callison-Burch (2011). Crowdsourcing translation: Professional
 12768 quality from non-professionals. In Proceedings of the Association for Computational
 12769 Linguistics (ACL), pp. 1220–1229.
- 12770 Zaremba, W., I. Sutskever, and O. Vinyals. Recurrent neural network regularization. arXiv
 12771 preprint arXiv:1409.2329.
- 12772 Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. arXiv preprint
 12773 arXiv:1212.5701.
- 12774 Zelenko, D., C. Aone, and A. Richardella (2003). Kernel methods for relation extraction.
 12775 The Journal of Machine Learning Research 3, 1083–1106.
- 12776 Zelle, J. M. and R. J. Mooney (1996). Learning to parse database queries using inductive
 12777 logic programming. In Proceedings of the National Conference on Artificial Intelligence
 12778 (AAAI), pp. 1050–1055.
- 12779 Zeng, D., K. Liu, S. Lai, G. Zhou, and J. Zhao (2014). Relation classification via
 12780 convolutional deep neural network. In Proceedings of the International Conference on
 12781 Computational Linguistics (COLING), pp. 2335–2344.
- 12782 Zettlemoyer, L. S. and M. Collins (2005). Learning to map sentences to logical form:
 12783 Structured classification with probabilistic categorial grammars. In Proceedings of UAI.
- 12784 Zhang, X., J. Zhao, and Y. LeCun (2015). Character-level convolutional networks for text
 12785 classification. In Neural Information Processing Systems (NIPS), pp. 649–657.
- 12786 Zhang, Y. and S. Clark (2008). A tale of two parsers: investigating and combining
 12787 graph-based and transition-based dependency parsing using beam-search. In Proceedings
 12788 of Empirical Methods for Natural Language Processing (EMNLP), pp. 562–571.
- 12789 Zhang, Y., T. Lei, R. Barzilay, T. Jaakkola, and A. Globerson (2014). Steps to excellence:
 12790 Simple inference with refined scoring of dependency trees. In Proceedings of the
 12791 Association for Computational Linguistics (ACL), pp. 197–207.
- 12792 Zhang, Y. and J. Nivre (2011). Transition-based dependency parsing with rich non-local
 12793 features. In Proceedings of the Association for Computational Linguistics (ACL), pp.
 12794 188–193.
- 12795 Zhou, J. and W. Xu (2015). End-to-end learning of semantic role labeling using recurrent
 12796 neural networks. In Proceedings of the Association for Computational Linguistics (ACL),
 12797 pp. 1127–1137.
- 12798 Zhu, J., Z. Nie, X. Liu, B. Zhang, and J.-R. Wen (2009). Statsnowball: a statistical
 12799 approach to extracting entity relationships. In Proceedings of the Conference on
 12800 World-Wide Web (WWW), pp. 101–110.

- 12801 Zhu, X., Z. Ghahramani, and J. D. Lafferty (2003). Semi-supervised learning using gaussian
12802 fields and harmonic functions. In Proceedings of the International Conference on Machine
12803 Learning (ICML), pp. 912–919.
- 12804 Zhu, X. and A. B. Goldberg (2009). Introduction to semi-supervised learning. *Synthesis*
12805 lectures on artificial intelligence and machine learning 3(1), 1–130.
- 12806 Zipf, G. K. (1949). Human behavior and the principle of least effort.
- 12807 Zirn, C., M. Niepert, H. Stuckenschmidt, and M. Strube (2011). Fine-grained sentiment
12808 analysis with structural features. In IJCNLP, Chiang Mai, Thailand, pp. 336–344.
- 12809 Zou, W. Y., R. Socher, D. Cer, and C. D. Manning (2013). Bilingual word embeddings
12810 for phrase-based machine translation. In Proceedings of Empirical Methods for Natural
12811 Language Processing (EMNLP), pp. 1393–1398.

₁₂₈₁₂ Index

- 12813 *K*-means, 108
12814 α -conversion, 305
12815 β -conversion, 303
12816 β -reduction, 303
12817 *n*-gram language models, 212
12818 *n*-gram, 39
12819 *F*-measure, 94
12820 bleu, 443
12821 WordNet, 86
- 12822 ablation test, 96
12823 absolute discounting, 142
12824 Abstract Meaning Representation
 (AMR), 317, 329
12825 abstractive summarization, 405, 472
12827 accepting path, 207
12828 accuracy, 38, 92
12829 action (reinforcement learning), 379
12830 active learning, 129
12831 AdaDelta (online optimization), 75
12832 AdaGrad (online optimization), 55, 74
12833 Adam (online optimization), 75
12834 adequacy (translation), 443
12835 adjectives, 191
12836 adjuncts (semantics), 316, 320
12837 adpositions, 192
12838 adverbs, 191
12839 adversarial networks, 125
12840 affix (morphology), 209
12841 agenda-based dialogue systems, 476
12842 agenda-based parsing, 290
- 12843 agent (thematic role), 318
12844 alignment, 331
12845 alignment (machine translation), 442,
 447
12847 alignment (text generation), 467
12848 Amazon Mechanical Turk, 102
12849 ambiguity, 224, 232
12850 ambiguity, attachment, 243
12851 ambiguity, complement structure, 243
12852 ambiguity, coordination scope, 243
12853 ambiguity, modifier scope, 243
12854 ambiguity, particle versus preposition,
 243
12855 anaphoric, 374
12857 anchored productions, 246
12858 animacy (semantics), 317
12859 annealing, 461
12860 antecedent (coreference), 365, 373
12861 antonymy, 86
12862 apophony, 208
12863 arc-eager dependency parsing, 281, 283
12864 arc-factored dependency parsing, 275
12865 arc-standard dependency parsing, 281
12866 area under the curve (auc), 95
12867 argumentation, 404
12868 argumentation mining, 404
12869 arguments, 413
12870 article (syntax), 196
12871 aspect, 191
12872 aspect-based opinion mining, 83
12873 attachment ambiguity, 269

- 12874 attention mechanism, 385, 436, 454, 468
 12875 autoencoder, 359
 12876 autoencoder, denoising, 359, 418
 12877 automated theorem provers, 298
 12878 automatic differentiation, 70
 12879 auxiliary verbs, 192
 12880 average mutual information, 346
 12881 averaged perceptron, 42
 12882 backchannel, 201
 12883 backoff, 142
 12884 backpropagation, 69, 148
 12885 backpropagation through time, 148
 12886 backward recurrence, 177, 178
 12887 backward-looking center, 394
 12888 bag of words, 29
 12889 balanced F -measure, 95
 12890 balanced test set, 93
 12891 batch learning, 40
 12892 batch normalization, 74
 12893 batch optimization, 53
 12894 Baum-Welch algorithm, 182
 12895 Bayes' rule, 484
 12896 Bayesian nonparametrics, 115, 262
 12897 beam sampling, 184
 12898 beam search, 284, 378, 459, 460
 12899 Bell number, 376
 12900 best-path algorithm, 212
 12901 bias (learning theory), 38, 139
 12902 bias-variance tradeoff, 38, 140
 12903 biconvexity, 113
 12904 bidirectional LSTM, 455
 12905 bidirectional recurrent neural network,
 180
 12907 bigrams, 39, 82
 12908 bilexical, 261
 12909 bilexical features, 278
 12910 bilinear product, 344
 12911 binarization (context-free grammar),
 225, 242
 12913 binomial distribution, 96
 12914 binomial random variable, 488
 12915 binomial test, 96
 12916 BIO notation, 198, 327
 12917 biomedical natural language processing,
 197
 12919 bipartite graph, 333
 12920 Bonferroni correction, 99
 12921 boolean semiring, 213
 12922 boosting, 62
 12923 bootstrap samples, 98
 12924 brevity penalty (machine translation),
 444
 12926 Brown clusters, 341, 345
 12927 byte-pair encoding, 357, 458
 12928 c-command, 367
 12929 case marking, 196, 233
 12930 Catalan number, 239
 12931 cataphora, 366
 12932 center embedding, 221
 12933 centering theory, 369, 394
 12934 chain FSA, 219
 12935 chain rule of probability, 483
 12936 chance agreement, 102
 12937 character-level language models, 153
 12938 chart parsing, 240
 12939 chatbots, 478
 12940 Chomsky Normal Form (CNF), 225
 12941 Chu-Liu-Edmonds algorithm, 276
 12942 CKY algorithm, 240
 12943 class imbalance, 93
 12944 classification weights, 29
 12945 closed-vocabulary, 153
 12946 closure (regular languages), 206
 12947 cloze question answering, 435
 12948 cluster ranking, 377
 12949 clustering, 108
 12950 co-training, 120
 12951 coarse-to-fine attention, 471
 12952 code switching, 194, 200
 12953 Cohen's Kappa, 102

- 12954 coherence, 408
 12955 cohesion, 391
 12956 collapsed Gibbs sampling, 128
 12957 collective entity linking, 418
 12958 collocation extraction, 358
 12959 collocation features, 87
 12960 combinatorial optimization, 19
 12961 combinatory categorial grammar, 233
 12962 complement clause, 227
 12963 complement event (probability), 481
 12964 composition (CCG), 234
 12965 compositional vector grammars, 403
 12966 compositionality, 18, 21, 355
 12967 computation graph, 62, 69
 12968 computational linguistics (versus natural language processing), 13
 12969 computational social science, 17
 12971 concept (AMR), 329
 12972 conditional independence, 166, 486
 12973 conditional log-likelihood, 66
 12974 conditional probability, 50, 483
 12975 conditional probability distribution, 487
 12976 conditional random field, 174
 12977 confidence interval, 98
 12978 configuration (transition-based parsing), 281
 12979 connected (graph theory), 331
 12981 consistency (logic), 301
 12982 constants (logic), 296
 12983 constituents, 226
 12984 constrained optimization, 47, 325
 12985 constraint-driven learning, 130
 12986 constraints, 325
 12987 content selection (text generation), 465
 12988 content words, 192
 12989 context vector (attentional neural translation), 456
 12990 context-free grammars (CFGs), 222
 12992 context-free languages, 221, 222
 12993 context-sensitive languages, 232
 12994 continuous bag-of-words (CBOW), 349
 12995 contradiction, 360
 12996 conversational turns, 201
 12997 convex optimization, 52
 12998 convexity, 44, 71, 310, 488, 491
 12999 convolutional neural networks, 67, 75, 82, 153, 182, 199, 425
 13000 cooperative principle, 365
 13002 coordinate ascent, 113
 13003 coordinating conjunctions, 192
 13004 coordinating discourse relations, 401
 13005 copula, 191, 231, 272
 13006 coreference resolution, 361, 365
 13007 coreferent, 365
 13008 cosine similarity, 353, 392
 13009 cost-augmented decoding, 49, 173
 13010 coverage (summarization), 405
 13011 coverage loss, 473
 13012 critical point, 72, 491
 13013 cross-document coreference resolution, 416
 13014 cross-entropy, 66, 426
 13016 cross-serial dependencies, 233
 13017 cross-validation, 39
 13018 crowdsourcing, 102
 13019 cumulative probability distribution, 97
 13020 dead neurons, 65
 13021 decidability (logic), 301
 13022 decision trees, 62
 13023 deep learning, 61
 13024 deep LSTM, 453
 13025 definiteness, 197
 13026 delta function, 37
 13027 denotation (semantics), 296
 13028 dependency grammar, 269
 13029 dependency graph, 270
 13030 dependency parse, 269
 13031 dependency path, 87, 324, 423
 13032 dependent, 270
 13033 derivation (context-free languages), 223
 13034 derivation (parsing), 280, 286

- 13035 derivation (semantic parsing), 304, 308
 13036 derivational ambiguity, 236
 13037 derivational morphology, 208
 13038 determiner, 193
 13039 determiner phrase, 229
 13040 deterministic FSA, 208
 13041 development set, 38, 92
 13042 dialogue acts, 102, 201, 480
 13043 dialogue management, 476
 13044 dialogue systems, 137, 475
 13045 digital humanities, 17, 81
 13046 dilated convolution, 77, 199
 13047 Dirichlet distribution, 127
 13048 discounting (language models), 142
 13049 discourse, 391
 13050 discourse connectives, 396
 13051 discourse depth, 405
 13052 discourse depth tree, 406, 407
 13053 discourse parsing, 396
 13054 discourse relations, 361, 396
 13055 discourse segment, 391
 13056 discourse sense classification, 398
 13057 discourse unit, 400
 13058 discrete random variable, 486
 13059 discriminative learning, 40
 13060 disjoint events (probability), 482
 13061 distant supervision, 130, 428, 429
 13062 distributed semantics, 341
 13063 distributional hypothesis, 339, 340
 13064 distributional semantics, 21, 341
 13065 distributional statistics, 87, 261, 340
 13066 document frequency, 417
 13067 domain adaptation, 107, 122
 13068 dropout, 71, 149
 13069 dual decomposition, 327
 13070 dynamic computation graphs, 71
 13071 dynamic oracle, 288
 13072 dynamic programming, 161
 13073 dynamic semantics, 313, 396
 13074 E-step (expectation-maximization), 111
 13075 early stopping, 42, 75
 13076 early update, 288
 13077 easy-first parsing, 290
 13078 edit distance, 215, 445
 13079 effective count (language models), 141
 13080 elementary discourse units, 400
 13081 elementwise nonlinearity, 63
 13082 Elman unit, 147
 13083 ELMo (embeddings from language models), 355
 13084 embedding, 179, 349
 13086 emission features, 160
 13087 emotion, 83
 13088 empirical Bayes, 128
 13089 empty string, 206
 13090 encoder-decoder, 452
 13091 encoder-decoder model, 359, 468
 13092 ensemble, 328
 13093 ensemble learning, 454
 13094 ensemble methods, 62
 13095 entailment, 301, 360
 13096 entities, 413
 13097 entity embeddings, 417
 13098 entity grid, 394
 13099 entity linking, 365, 413, 415, 426
 13100 entropy, 56, 111
 13101 estimation, 488
 13102 EuroParl corpus, 445
 13103 event, 430
 13104 event (probability), 481
 13105 event coreference, 431
 13106 event detection, 430
 13107 event semantics, 315
 13108 events, 413
 13109 evidentiality, 196, 433
 13110 exchange clustering, 348
 13111 expectation, 487
 13112 expectation maximization, 109, 143
 13113 expectation semiring, 221
 13114 expectation-maximization, in machine translation, 449
 13115

- 13116 explicit semantic analysis, 342
 13117 exploding gradients, 149
 13118 extra-propositional semantics, 432
 13119 extractive question-answering, 435
 13120 extractive summarization, 405
 13121 extrinsic evaluation, 151, 353
 13122 factoid questions, 332
 13123 factoids, 434
 13124 factor graph, 175
 13125 factuality, 433
 13126 false discovery rate, 99
 13127 False negative, 93
 13128 False positive, 93
 13129 false positive, 485
 13130 false positive rate, 95, 484
 13131 feature co-adaptation, 71
 13132 feature function, 30, 39
 13133 feature hashing, 92
 13134 feature noising, 71
 13135 feature selection, 55
 13136 features, 18
 13137 feedforward neural network, 64
 13138 fine-tuned word embeddings, 355
 13139 finite state acceptor (FSA), 207
 13140 finite state automata, 207
 13141 finite state composition, 218
 13142 finite state transducers, 210, 215
 13143 first-order logic, 298
 13144 fluency (translation), 443
 13145 fluent, 137
 13146 focus, 330
 13147 formal language theory, 205
 13148 forward recurrence, 176
 13149 forward variable, 178
 13150 forward variables, 176
 13151 forward-backward algorithm, 177, 220, 253
 13152 forward-looking centers, 394
 13153 frame, 475
 13154 frame elements, 320
 13156 FrameNet, 320
 13157 frames, 320
 13158 Frobenius norm, 71
 13159 function (first-order logic), 300
 13160 function words, 192
 13161 functional margin, 46
 13162 functional segmentation, 391, 393
 13163 garden path sentence, 158
 13164 gate (neural networks), 66, 455
 13165 gazetteer, 423
 13166 gazetteers, 371
 13167 gazzeteers, 198
 13168 generalization, 42
 13169 generalized linear models, 56
 13170 generative model, 33, 249
 13171 generative models, 377
 13172 generative process, 143
 13173 generic referents, 370
 13174 geometric margin, 46
 13175 Gibbs sampling, 127, 419
 13176 gloss, 137, 193, 443
 13177 government and binding theory, 367
 13178 gradient, 44
 13179 gradient clipping, 74
 13180 gradient descent, 53
 13181 Gram matrix, 423
 13182 grammar induction, 255
 13183 grammaticality, 408
 13184 graph-based dependency parsing, 274
 13185 graphical model, 166
 13186 graphics processing units (GPUs), 182, 199
 13187 grid search, 38
 13188 Hamming cost, 173
 13189 Hansards corpus, 445
 13190 hanzi, 89
 13192 hard expectation-maximization, 114
 13193 head, 270, 423
 13194 head percolation rules, 258

- 13195 head rules, 269
 13196 head word, 226, 269, 371
 13197 head words, 258
 13198 hedging, 433
 13199 held-out data, 151
 13200 Hessian matrix, 53
 13201 hidden Markov models, 166
 13202 hidden variable perceptron, 221
 13203 hierarchical clustering, 345
 13204 hierarchical recurrent network, 478
 13205 hierarchical softmax, 147, 351
 13206 hierarchical topic segmentation, 392
 13207 highway network, 66
 13208 hinge loss, 44
 13209 holonymy, 86
 13210 homonym, 85
 13211 human computation, 103
 13212 hypergraph, 404
 13213 hyponymy, 86
 13214 hyperparameter, 38
 13215 hyponymy, 86
 13216 illocutionary force, 201
 13217 implicit discourse relations, 398
 13218 importance sampling, 462
 13219 importance score, 462
 13220 incremental expectation maximization, 114
 13221 incremental perceptron, 288, 378
 13222 independent and identically distributed (IID), 32
 13223 indicator function, 37
 13224 indicator random variable, 486
 13225 inference, 159
 13226 inference (logic), 295
 13227 inference rules, 298
 13228 inflection point, 492
 13229 inflectional affixes, 90
 13230 inflectional morphology, 191, 208, 216
 13231 information extraction, 413
 13232 information retrieval, 17, 426
 13235 initiative (dialogue systems), 476
 13236 input word embeddings, 147
 13237 inside recurrence, 249, 250, 253
 13238 inside-outside algorithm, 253, 262
 13239 instance (AMR), 329
 13240 instance labels, 32
 13241 integer linear program, 438, 474
 13242 integer linear programming, 325, 376, 406, 419
 13244 inter-annotator agreement, 102
 13245 interjections, 191
 13246 interlingua, 442
 13247 interpolated n -gram language model, 213
 13248 interpolation, 143
 13249 interval algebra, 431
 13250 intrinsic evaluation, 151, 353
 13251 inverse document frequency, 417
 13252 inverse relation (AMR), 330
 13253 inversion (finite state), 217
 13254 irrealis, 82
 13255 Jeffreys-Perks law, 141
 13256 Jensen's inequality, 110
 13257 joint probabilities, 487
 13258 joint probability, 32, 50
 13259 Kalman smoother, 184
 13260 Katz backoff, 142
 13261 kernel function, 423
 13262 kernel methods, 62
 13263 kernel support vector machine, 62, 424
 13264 Kleene star, 206
 13265 knapsack problem, 406
 13266 knowledge base, 413
 13267 knowledge base population, 426
 13268 L-BFGS, 53
 13269 label bias, 291
 13270 label bias problem, 287
 13271 label propagation, 121, 132
 13272 labeled dependencies, 271
 13273 labeled precision, 244

- 13274 labeled recall, 244
 13275 Lagrange multiplier, 493
 13276 Lagrangian, 493
 13277 Lagrangian dual, 493
 13278 lambda calculus, 302
 13279 lambda expressions, 303
 13280 language model, 138
 13281 language models, 16
 13282 Laplace smoothing, 38, 141
 13283 large margin classification, 45
 13284 latent conditional random fields, 311
 13285 latent Dirichlet allocation, 392
 13286 latent semantic analysis, 342, 344
 13287 latent variable, 110, 220, 309, 428, 442
 13288 latent variable perceptron, 311, 374
 13289 layer normalization, 74, 457
 13290 leaky ReLU, 65
 13291 learning to search, 269, 288, 380
 13292 least squares, 84
 13293 leave-one-out, 39
 13294 lemma, 85
 13295 lemma (lexical semantics), 216
 13296 lemmatization, 90
 13297 Levenshtein edit distance, 215
 13298 lexical entry, 304
 13299 lexical features, 61
 13300 lexical semantics, 85
 13301 lexical unit (frame semantics), 320
 13302 lexicalization, 258
 13303 lexicalization (text generation), 465
 13304 lexicalized tree-adjoining grammar for discourse (D-LTAG), 397
 13305 lexicon, 304
 13307 lexicon (CCG), 234
 13308 lexicon-based classification, 84
 13309 lexicon-based sentiment analysis, 82
 13310 Lidstone smoothing, 141
 13311 light verb, 331
 13312 likelihood, 484
 13313 linear regression, 84
 13314 linear separability, 41
 13315 linearization, 475
 13316 literal character, 206
 13317 local minimum, 492
 13318 local optimum, 113
 13319 locally-normalized objective, 287
 13320 log-bilinear language model, 357
 13321 logistic function, 56
 13322 logistic loss, 50
 13323 logistic regression, 49, 56
 13324 Long short-term memories, 147
 13325 long short-term memory, 149
 13326 long short-term memory (LSTM), 66, 195, 452
 13328 lookup layer, 67, 147
 13329 loss function, 42
 13330 LSTM, 149
 13331 LSTM-CRF, 181, 328
 13332 machine learning, 14
 13333 machine reading, 434
 13334 machine translation, 137
 13335 Macro *F*-measure, 94
 13336 macro-reading, 414
 13337 margin, 41, 45
 13338 marginal probability distribution, 487
 13339 marginal relevance, 473
 13340 marginalize, 483
 13341 markable, 372
 13342 Markov assumption, 166
 13343 Markov blanket, 166
 13344 Markov Chain Monte Carlo (MCMC), 115, 126, 184
 13345
 13346 Markov decision process, 476
 13347 Markov random fields, 175
 13348 matrix-tree theorem, 280
 13349 max-margin Markov network, 173
 13350 max-product algorithm, 169
 13351 maximum a posteriori, 38, 489
 13352 maximum conditional likelihood, 50
 13353 maximum entropy, 56
 13354 maximum likelihood, 488

- 13355 maximum likelihood estimate, 36
 13356 maximum likelihood estimation, 32
 13357 maximum spanning tree, 276
 13358 McNemar’s test, 96
 13359 meaning representation, 295
 13360 membership problem, 205
 13361 memory cell (LSTM), 149
 13362 mention (coreference resolution), 365
 13363 mention (entity), 413
 13364 mention (information extraction), 415
 13365 mention ranking, 374
 13366 mention-pair model, 373
 13367 meronymy, 86
 13368 meteor, 445
 13369 method of moments, 128
 13370 micro *F*-measure, 94
 13371 micro-reading, 414
 13372 mildly context-sensitive languages, 233
 13373 minibatch, 54
 13374 minimization (FSA), 210
 13375 minimum error-rate training (MERT),
 461
 13377 minimum risk training, 461
 13378 mixed-initiative, 476
 13379 modality, 432
 13380 model, 19
 13381 model builder, 301
 13382 model checker, 301
 13383 model-theoretic semantics, 296
 13384 modeling (machine learning), 52
 13385 modifier (dependency grammar), 270
 13386 modus ponens, 298
 13387 moment-matching, 56
 13388 monomorphemic, 210
 13389 morphemes, 17, 153, 209
 13390 morphological analysis, 216
 13391 morphological generation, 216
 13392 morphological segmentation, 171
 13393 morphology, 91, 170, 208, 356, 457
 13394 morphosyntactic, 190
 13395 morphosyntactic attributes, 194
 13396 morphotactic, 209
 13397 multi-document summarization, 474
 13398 multi-view learning, 120
 13399 multilayer perceptron, 64
 13400 multinomial distribution, 33
 13401 multinomial naïve Bayes, 34
 13402 multiple instance learning, 130, 428
 13403 multiple-choice question answering, 435
 13404 multitask learning, 130
 13405 Naïve Bayes, 33
 13406 name dictionary, 416
 13407 named entities, 197
 13408 named entity linking, 415
 13409 named entity recognition, 181, 413, 415
 13410 named entity types, 415
 13411 narrow convolution, 76
 13412 nearest-neighbor, 62, 424
 13413 negation, 82, 432
 13414 negative sampling, 351, 352, 421
 13415 Neo-Davidsonian event semantics, 316
 13416 neural attention, 453
 13417 neural machine translation, 442
 13418 neural networks, 61, 145
 13419 NIL entity, 415
 13420 noise-contrastive estimation, 147
 13421 noisy channel model, 138, 446
 13422 nominal modifier, 229
 13423 nominals, 365
 13424 nominals (coreference), 372
 13425 non-core roles (AMR), 330
 13426 non-terminals (context-free grammars),
 223
 13428 normalization, 90
 13429 noun phrase, 14, 226
 13430 nouns, 190
 13431 NP-hard, 55, 419
 13432 nuclearity (RST), 401
 13433 nucleus (RST), 401
 13434 null hypothesis, 96
 13435 numeral (part of speech), 193

- 13436 numerical optimization, 20
 13437 offset feature, 31
 13438 one-dimensional convolution, 76
 13439 one-hot, 386
 13440 one-hot vector, 66
 13441 one-tailed p-value, 97
 13442 one-versus-all multiclass classification, 424
 13443 one-versus-one multiclass classification, 424
 13444 online expectation maximization, 114
 13445 online learning, 40, 54
 13446 ontology, 20
 13447 open information extraction, 429
 13448 open word classes, 190
 13449 opinion polarity, 81
 13450 oracle, 286, 332
 13451 oracle (learning to search), 380
 13452 orthography, 210, 218
 13453 orthonormal matrix, 73
 13454 out-of-vocabulary words, 195
 13455 outside recurrence, 250, 254
 13456 overfit, 38
 13457 overfitting, 42
 13458 overgeneration, 217, 227
 13459 parallel corpora, 445
 13460 parameters, 488
 13461 paraphrase, 360
 13462 parent annotation, 257
 13463 parsing, 223
 13464 part-of-speech, 189
 13465 part-of-speech tagging, 157
 13466 partially observable Markov decision process (POMDP), 478
 13467 partially supervised learning, 262
 13468 particle (part-of-speech), 193, 231
 13469 partition, 483
 13470 partition function, 176
 13471 parts-of-speech, 17
 13472 passive-aggressive, 493
 13473 path (finite state automata), 207
 13474 Penn Discourse Treebank (PDTB), 397
 13475 Penn Treebank, 152, 171, 194, 226, 252
 13476 perceptron, 40
 13477 perplexity, 152
 13478 phonology, 210
 13479 phrase (syntax), 226
 13480 phrase-structure grammar, 226
 13481 pivot features, 124
 13482 pointwise mutual information, 343, 344
 13483 policy, 379, 477
 13484 policy (search), 288
 13485 policy gradient, 380
 13486 polysemous, 86
 13487 pooling, 387
 13488 pooling (convolution), 76, 386, 468
 13489 positional encodings, 457
 13490 positive pointwise mutual information, 345
 13491 posterior, 484
 13492 power law, 14
 13493 pragmatics, 365
 13494 pre-trained word representations, 354
 13495 precision, 94, 484
 13496 precision-at- k , 95, 409
 13497 precision-recall curve, 428
 13498 precision-recall curves, 95
 13499 predicate, 413
 13500 predicative adjectives, 231
 13501 predictive likelihood, 115
 13502 prepositional phrase, 14, 231
 13503 primal form, 493
 13504 principle of compositionality, 302
 13505 prior, 484
 13506 prior expectation, 489
 13507 probabilistic context-free grammars (PCFGs), 249
 13508 probabilistic models, 488
 13509 probabilistic topic model, 419
 13510 probability density function, 487

- 13516 probability distribution, 486
 13517 probability mass function, 97, 486
 13518 probability simplex, 33
 13519 processes, 431
 13520 production rules, 223
 13521 productivity, 209
 13522 projection function, 124
 13523 projectivity, 273
 13524 pronominal anaphora resolution, 365
 13525 pronoun, 192
 13526 PropBank, 320
 13527 proper nouns, 191
 13528 property (logic), 299
 13529 proposal distribution, 462
 13530 proposition, 432
 13531 propositions, 296, 297
 13532 prosody, 201
 13533 proto-roles, 319
 13534 pseudo-projective dependency parsing,
 284
 13536 pumping lemma, 221
 13537 pushdown automata, 223
 13538 pushdown automaton, 264
 13539 quadratic program, 47
 13540 quantifier, 299
 13541 quantifier, existential, 300
 13542 quantifier, universal, 300
 13543 quasi-Newton optimization, 53
 13544 question answering, 360, 415
 13545 random outcomes, 481
 13546 ranking, 416
 13547 ranking loss, 416
 13548 recall, 94, 484
 13549 recall-at- k , 409
 13550 receiver operating characteristic (ROC),
 95
 13552 rectified linear unit (ReLU), 65
 13553 recurrent neural network, 146
 13554 recurrent neural networks, 425
 13555 recursion, 14
 13556 recursive neural network, 407
 13557 recursive neural networks, 263, 357, 360
 13558 recursive production, 223
 13559 reference arguments, 335
 13560 reference resolution, 365
 13561 reference translations, 443
 13562 referent, 365
 13563 referring expression, 394
 13564 referring expressions, 365, 466
 13565 reflexive pronoun, 367
 13566 regression, 84
 13567 regular expression, 206
 13568 regular language, 206
 13569 regularization, 49
 13570 reification (events), 315
 13571 reinforcement learning, 379, 460
 13572 relation (logic), 305
 13573 relation extraction, 289, 333, 421
 13574 relations, 413
 13575 relations (information extraction), 413
 13576 relations (logic), 296
 13577 relative frequency estimate, 36, 138, 489
 13578 reranking, 263
 13579 residual networks, 66
 13580 retrofitting (word embeddings), 357
 13581 Rhetorical Structure Theory (RST), 400
 13582 rhetorical zones, 393
 13583 RIBES (translation metric), 445
 13584 ridge regression, 84
 13585 risk, 461
 13586 roll-in (reinforcement learning), 380
 13587 roll-out (reinforcement learning), 380
 13588 root (morpheme), 357
 13589 saddle point, 492
 13590 saddle points, 72
 13591 sample space, 481
 13592 satellite (RST), 401
 13593 satisfaction (logic), 301
 13594 scheduled sampling, 460

- 13595 schema, 413, 414, 429
 13596 search error, 265, 377
 13597 second-order dependency parsing, 275
 13598 second-order logic, 299
 13599 seed lexicon, 85
 13600 segmented discourse representation theory (SDRT), 396
 13601 self-attention, 456
 13603 self-training, 120
 13604 semantic, 190
 13605 semantic concordance, 87
 13606 semantic parsing, 302
 13607 semantic role, 316
 13608 Semantic role labeling, 316
 13609 semantic role labeling, 421, 429
 13610 semantic underspecification, 312
 13611 semantics, 256, 295
 13612 semi-supervised learning, 107, 117, 354
 13613 semiring algebra, 184
 13614 semiring notation, 213
 13615 semisupervised, 88
 13616 senses, 319
 13617 sentence (logic), 300
 13618 sentence compression, 473
 13619 sentence fusion, 474
 13620 sentence summarization, 472
 13621 sentiment, 81
 13622 sentiment lexicon, 32
 13623 sequence-to-sequence, 452
 13624 shift-reduce parsing, 264
 13625 shifted positive pointwise mutual information, 352
 13626 shortest-path algorithm, 211
 13628 sigmoid, 63
 13629 simplex, 127
 13630 singular value decomposition, 73
 13631 singular value decomposition (SVD), 116
 13632 singular vectors, 73
 13633 skipgram word embeddings, 350
 13634 slack variables, 48
 13635 slot filling, 426
 13636 slots (dialogue systems), 475
 13637 smooth functions, 45
 13638 smoothing, 38, 141
 13639 soft K -means, 109
 13640 softmax, 63, 146, 426
 13641 source domain, 122
 13642 source language, 441
 13643 spanning tree, 270
 13644 sparse matrix, 344
 13645 sparsity, 55, 56
 13646 spectral learning, 129
 13647 speech acts, 201
 13648 speech recognition, 137
 13649 split constituents, 322
 13650 spurious ambiguity, 236, 280, 286, 308
 13651 squashing function, 147
 13652 squashing functions, 65
 13653 stand-off annotations, 101
 13654 Stanford Natural Language Inference corpus, 360
 13655 statistical learning theory, 41
 13657 statistical machine translation, 442
 13658 statistical significance, 96
 13659 stem, 18
 13660 stem (morphology), 209
 13661 stemmer, 90
 13662 step size, 53, 492
 13663 stochastic gradient descent, 44, 54
 13664 stoplist, 92
 13665 stopwords, 91
 13666 string (formal language theory), 205
 13667 string-to-tree translation, 451
 13668 strong compositionality criterion (RST), 403
 13669
 13670 strongly equivalent grammars, 224
 13671 structure induction, 182
 13672 structured attention, 471
 13673 structured perceptron, 172
 13674 structured prediction, 30
 13675 structured support vector machine, 173
 13676 subgradient, 45, 56

- 13677 subjectivity detection, 82
 13678 subordinating conjunctions, 192
 13679 subordinating discourse relations, 401
 13680 sum-product algorithm, 176
 13681 summarization, 137, 405
 13682 supersenses, 354
 13683 supervised machine learning, 32
 13684 support vector machine, 48
 13685 support vectors, 48
 13686 surface form, 217
 13687 surface realization, 465
 13688 synchronous context-free grammar, 451
 13689 synonymy, 86, 339
 13690 synset (synonym set), 86
 13691 synsets, 383
 13692 syntactic dependencies, 270
 13693 syntactic path, 323
 13694 syntactic-semantic grammar, 303
 13695 syntax, 189, 225, 295
- 13696 tagset, 190
 13697 tanh activation function, 65
 13698 target domain, 122
 13699 target language, 441
 13700 Targeted sentiment analysis, 83
 13701 tense, 191
 13702 terminal symbols (context-free grammars), 223
 13703 test set, 38, 107
 13705 test statistic, 96
 13706 text classification, 29
 13707 text mining, 17
 13708 text planning, 465
 13709 thematic roles, 317
 13710 third axiom of probability, 482
 13711 third-order dependency parsing, 276
 13712 TimeML, 431
 13713 tokenization, 88, 199
 13714 tokens, 34
 13715 topic models, 17
 13716 topic segmentation, 391
- 13717 trace (syntax), 236
 13718 training set, 32, 107
 13719 transduction, 206
 13720 transfer learning, 130
 13721 transformer architecture, 456
 13722 transition features, 160
 13723 transition system, 281
 13724 transition-based parsing, 239, 264
 13725 transitive closure, 375
 13726 translation error rate (TER), 445
 13727 translation model, 138
 13728 transliteration, 458
 13729 tree-adjoining grammar, 233
 13730 tree-to-string translation, 452
 13731 tree-to-tree translation, 451
 13732 treebank, 252
 13733 trellis, 162, 219
 13734 trigrams, 39
 13735 trilexical dependencies, 261
 13736 tropical semiring, 185, 213
 13737 True negative, 93
 13738 True positive, 93
 13739 true positive, 485
 13740 true positive rate, 95
 13741 truncated singular value decomposition, 344
 13742 truth conditions, 300
 13744 tuning set, 38, 92
 13745 Turing test, 15
 13746 two-tailed test, 97
 13747 type systems, 306
 13748 type-raising, 234, 306
 13749 types, 34
- 13750 unary closure, 243
 13751 unary productions, 224
 13752 underfit, 38
 13753 underflow, 32
 13754 undergeneration, 217, 227
 13755 Universal Dependencies, 190, 269
 13756 unlabeled precision, 244

- 13757 unlabeled recall, 244
13758 unseen word, 181
13759 unsupervised, 88
13760 unsupervised learning, 83, 107
13761 utterances, 201
13762 validation function (semantic parsing),
 312
13764 validity (logic), 301
13765 value function, 477
13766 value iteration, 477
13767 vanishing gradient, 65
13768 vanishing gradients, 149
13769 variable (AMR), 329
13770 variable (logic), 299
13771 variable, bound (logic), 300
13772 variable, free (logic), 299
13773 variance, 99
13774 variance (learning theory), 37
13775 variational autoencoder, 473
13776 Vauquois Pyramid, 442
13777 verb phrase, 227
13778 VerbNet, 318
13779 verbs, 191
13780 vertical Markovization, 257
13781 Viterbi algorithm, 160
13782 Viterbi variable, 162
13783 volition (semantics), 317
13784 WARP loss, 421
13785 weakly equivalent grammars, 224
13786 weight decay, 71
13787 weighted context-free grammar, 246
13788 weighted context-free grammars, 232,
 242
13790 weighted finite state acceptors, 211
13791 wide convolution, 76
13792 Wikification, 415
13793 Winograd schemas, 15
13794 word embedding, 67
13795 word embeddings, 61, 148, 341
13796 word representations, 340
13797 word sense disambiguation, 85
13798 word senses, 85
13799 word tokens, 88
13800 WordNet, 20
13801 world model, 296
13802 yield (context-free grammars), 223
13803 zero-one loss, 44
13804 Zipf's law, 155