

1

# Natural Language Processing

2

Jacob Eisenstein

3

July 10, 2018



# **4** Contents

<b>5</b>	<b>Contents</b>	<b>1</b>
<b>6</b>	<b>1 Introduction</b>	<b>13</b>
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
<b>15</b>	<b>I Learning</b>	<b>27</b>
<b>16</b>	<b>2 Linear text classification</b>	<b>29</b>
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	<b>3 Nonlinear classification</b>	<b>61</b>
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	<b>4 Linguistic applications of classification</b>	<b>81</b>
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 <i>F</i> -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	<b>5 Learning without supervision</b>	<b>107</b>
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	<b>II Sequences and trees</b>	<b>135</b>
92	<b>6 Language models</b>	<b>137</b>
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	<b>7 Sequence labeling</b>	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	<b>8 Applications of sequence labeling</b>	187
129	8.1 Part-of-speech tagging . . . . .	187
130	8.1.1 Parts-of-Speech . . . . .	188
131	8.1.2 Accurate part-of-speech tagging . . . . .	192
132	8.2 Morphosyntactic Attributes . . . . .	194
133	8.3 Named Entity Recognition . . . . .	195
134	8.4 Tokenization . . . . .	197
135	8.5 Code switching . . . . .	198
136	8.6 Dialogue acts . . . . .	199
137	<b>9 Formal language theory</b>	201
138	9.1 Regular languages . . . . .	202
139	9.1.1 Finite state acceptors . . . . .	203
140	9.1.2 Morphology as a regular language . . . . .	204
141	9.1.3 Weighted finite state acceptors . . . . .	206
142	9.1.4 Finite state transducers . . . . .	211
143	9.1.5 *Learning weighted finite state automata . . . . .	216
144	9.2 Context-free languages . . . . .	217
145	9.2.1 Context-free grammars . . . . .	218

146	9.2.2 Natural language syntax as a context-free language . . . . .	221
147	9.2.3 A phrase-structure grammar for English . . . . .	223
148	9.2.4 Grammatical ambiguity . . . . .	228
149	9.3 *Mildly context-sensitive languages . . . . .	228
150	9.3.1 Context-sensitive phenomena in natural language . . . . .	229
151	9.3.2 Combinatory categorial grammar . . . . .	230
152	<b>10 Context-free parsing</b>	<b>235</b>
153	10.1 Deterministic bottom-up parsing . . . . .	236
154	10.1.1 Recovering the parse tree . . . . .	238
155	10.1.2 Non-binary productions . . . . .	238
156	10.1.3 Complexity . . . . .	239
157	10.2 Ambiguity . . . . .	239
158	10.2.1 Parser evaluation . . . . .	240
159	10.2.2 Local solutions . . . . .	241
160	10.3 Weighted Context-Free Grammars . . . . .	242
161	10.3.1 Parsing with weighted context-free grammars . . . . .	243
162	10.3.2 Probabilistic context-free grammars . . . . .	245
163	10.3.3 *Semiring weighted context-free grammars . . . . .	247
164	10.4 Learning weighted context-free grammars . . . . .	247
165	10.4.1 Probabilistic context-free grammars . . . . .	248
166	10.4.2 Feature-based parsing . . . . .	248
167	10.4.3 *Conditional random field parsing . . . . .	249
168	10.4.4 Neural context-free grammars . . . . .	251
169	10.5 Grammar refinement . . . . .	252
170	10.5.1 Parent annotations and other tree transformations . . . . .	253
171	10.5.2 Lexicalized context-free grammars . . . . .	254
172	10.5.3 *Refinement grammars . . . . .	258
173	10.6 Beyond context-free parsing . . . . .	259
174	10.6.1 Reranking . . . . .	259
175	10.6.2 Transition-based parsing . . . . .	260
176	<b>11 Dependency parsing</b>	<b>263</b>
177	11.1 Dependency grammar . . . . .	263
178	11.1.1 Heads and dependents . . . . .	264
179	11.1.2 Labeled dependencies . . . . .	265
180	11.1.3 Dependency subtrees and constituents . . . . .	266
181	11.2 Graph-based dependency parsing . . . . .	268
182	11.2.1 Graph-based parsing algorithms . . . . .	270
183	11.2.2 Computing scores for dependency arcs . . . . .	271
184	11.2.3 Learning . . . . .	273

185	11.3 Transition-based dependency parsing . . . . .	274
186	11.3.1 Transition systems for dependency parsing . . . . .	275
187	11.3.2 Scoring functions for transition-based parsers . . . . .	279
188	11.3.3 Learning to parse . . . . .	280
189	11.4 Applications . . . . .	283
190	<b>III Meaning</b>	<b>287</b>
191	<b>12 Logical semantics</b>	<b>289</b>
192	12.1 Meaning and denotation . . . . .	290
193	12.2 Logical representations of meaning . . . . .	291
194	12.2.1 Propositional logic . . . . .	291
195	12.2.2 First-order logic . . . . .	292
196	12.3 Semantic parsing and the lambda calculus . . . . .	296
197	12.3.1 The lambda calculus . . . . .	297
198	12.3.2 Quantification . . . . .	299
199	12.4 Learning semantic parsers . . . . .	301
200	12.4.1 Learning from derivations . . . . .	302
201	12.4.2 Learning from logical forms . . . . .	304
202	12.4.3 Learning from denotations . . . . .	305
203	<b>13 Predicate-argument semantics</b>	<b>311</b>
204	13.1 Semantic roles . . . . .	313
205	13.1.1 VerbNet . . . . .	314
206	13.1.2 Proto-roles and PropBank . . . . .	315
207	13.1.3 FrameNet . . . . .	316
208	13.2 Semantic role labeling . . . . .	318
209	13.2.1 Semantic role labeling as classification . . . . .	318
210	13.2.2 Semantic role labeling as constrained optimization . . . . .	321
211	13.2.3 Neural semantic role labeling . . . . .	323
212	13.3 Abstract Meaning Representation . . . . .	324
213	13.3.1 AMR Parsing . . . . .	327
214	13.4 Applications of Predicate-Argument Semantics . . . . .	328
215	<b>14 Distributional and distributed semantics</b>	<b>335</b>
216	14.1 The distributional hypothesis . . . . .	335
217	14.2 Design decisions for word representations . . . . .	337
218	14.2.1 Representation . . . . .	337
219	14.2.2 Context . . . . .	338
220	14.2.3 Estimation . . . . .	339

221	14.3 Latent semantic analysis . . . . .	340
222	14.4 Brown clusters . . . . .	341
223	14.5 Neural word embeddings . . . . .	345
224	14.5.1 Continuous bag-of-words (CBOW) . . . . .	345
225	14.5.2 Skipgrams . . . . .	346
226	14.5.3 Computational complexity . . . . .	346
227	14.5.4 Word embeddings as matrix factorization . . . . .	348
228	14.6 Evaluating word embeddings . . . . .	349
229	14.6.1 Intrinsic evaluations . . . . .	349
230	14.6.2 Extrinsic evaluations . . . . .	350
231	14.7 Distributed representations beyond distributional statistics . . . . .	351
232	14.7.1 Word-internal structure . . . . .	352
233	14.7.2 Lexical semantic resources . . . . .	354
234	14.8 Distributed representations of multiword units . . . . .	354
235	14.8.1 Purely distributional methods . . . . .	354
236	14.8.2 Distributional-compositional hybrids . . . . .	355
237	14.8.3 Supervised compositional methods . . . . .	356
238	14.8.4 Hybrid distributed-symbolic representations . . . . .	357
239	<b>15 Reference Resolution</b>	<b>361</b>
240	15.1 Forms of referring expressions . . . . .	362
241	15.1.1 Pronouns . . . . .	362
242	15.1.2 Proper Nouns . . . . .	367
243	15.1.3 Nominals . . . . .	368
244	15.2 Algorithms for coreference resolution . . . . .	368
245	15.2.1 Mention-pair models . . . . .	369
246	15.2.2 Mention-ranking models . . . . .	370
247	15.2.3 Transitive closure in mention-based models . . . . .	371
248	15.2.4 Entity-based models . . . . .	372
249	15.3 Representations for coreference resolution . . . . .	377
250	15.3.1 Features . . . . .	378
251	15.3.2 Distributed representations of mentions and entities . . . . .	380
252	15.4 Evaluating coreference resolution . . . . .	383
253	<b>16 Discourse</b>	<b>387</b>
254	16.1 Segments . . . . .	387
255	16.1.1 Topic segmentation . . . . .	388
256	16.1.2 Functional segmentation . . . . .	389
257	16.2 Entities and reference . . . . .	389
258	16.2.1 Centering theory . . . . .	390
259	16.2.2 The entity grid . . . . .	391

260	16.2.3 *Formal semantics beyond the sentence level . . . . .	392
261	16.3 Relations . . . . .	392
262	16.3.1 Shallow discourse relations . . . . .	393
263	16.3.2 Hierarchical discourse relations . . . . .	396
264	16.3.3 Argumentation . . . . .	400
265	16.3.4 Applications of discourse relations . . . . .	401
266	<b>IV Applications</b>	<b>407</b>
267	<b>17 Information extraction</b>	<b>409</b>
268	17.1 Entities . . . . .	411
269	17.1.1 Entity linking by learning to rank . . . . .	412
270	17.1.2 Collective entity linking . . . . .	414
271	17.1.3 *Pairwise ranking loss functions . . . . .	415
272	17.2 Relations . . . . .	417
273	17.2.1 Pattern-based relation extraction . . . . .	418
274	17.2.2 Relation extraction as a classification task . . . . .	419
275	17.2.3 Knowledge base population . . . . .	422
276	17.2.4 Open information extraction . . . . .	426
277	17.3 Events . . . . .	427
278	17.4 Hedges, denials, and hypotheticals . . . . .	428
279	17.5 Question answering and machine reading . . . . .	430
280	17.5.1 Formal semantics . . . . .	430
281	17.5.2 Machine reading . . . . .	431
282	<b>18 Machine translation</b>	<b>437</b>
283	18.1 Machine translation as a task . . . . .	437
284	18.1.1 Evaluating translations . . . . .	439
285	18.1.2 Data . . . . .	441
286	18.2 Statistical machine translation . . . . .	442
287	18.2.1 Statistical translation modeling . . . . .	443
288	18.2.2 Estimation . . . . .	445
289	18.2.3 Phrase-based translation . . . . .	446
290	18.2.4 *Syntax-based translation . . . . .	447
291	18.3 Neural machine translation . . . . .	448
292	18.3.1 Neural attention . . . . .	450
293	18.3.2 *Neural machine translation without recurrence . . . . .	452
294	18.3.3 Out-of-vocabulary words . . . . .	453
295	18.4 Decoding . . . . .	455
296	18.5 Training towards the evaluation metric . . . . .	457

297	<b>19 Text generation</b>	<b>461</b>
298	19.1 Data-to-text generation . . . . .	461
299	19.1.1 Latent data-to-text alignment . . . . .	463
300	19.1.2 Neural data-to-text generation . . . . .	464
301	19.2 Text-to-text generation . . . . .	468
302	19.2.1 Neural abstractive summarization . . . . .	468
303	19.2.2 Sentence fusion for multi-document summarization . . . . .	470
304	19.3 Dialogue . . . . .	471
305	19.3.1 Finite-state and agenda-based dialogue systems . . . . .	471
306	19.3.2 Markov decision processes . . . . .	472
307	19.3.3 Neural chatbots . . . . .	474
308	<b>A Probability</b>	<b>477</b>
309	A.1 Probabilities of event combinations . . . . .	477
310	A.1.1 Probabilities of disjoint events . . . . .	478
311	A.1.2 Law of total probability . . . . .	479
312	A.2 Conditional probability and Bayes' rule . . . . .	479
313	A.3 Independence . . . . .	481
314	A.4 Random variables . . . . .	482
315	A.5 Expectations . . . . .	483
316	A.6 Modeling and estimation . . . . .	484
317	<b>B Numerical optimization</b>	<b>487</b>
318	B.1 Gradient descent . . . . .	488
319	B.2 Constrained optimization . . . . .	488
320	B.3 Example: Passive-aggressive online learning . . . . .	489
321	<b>Bibliography</b>	<b>491</b>



# 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 ( $\alpha, \beta, \theta$ ). Vectors are  
325 indicated with bold script for both random variables  $\mathbf{x}$  and parameters  $\boldsymbol{\theta}$ . Other useful  
326 notations are indicated in the table below.

---

## Basics

---

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

---

## Linear algebra

---

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

---

**Text datasets**

---

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

---

**Probabilities**

---

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

---

**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 $y$
$\theta$	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
$\lambda$	the amount of regularization

327 **Chapter 1**

328 **Introduction**

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

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

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

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

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

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

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

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

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

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

---

<sup>1</sup>Throughout the text, this notation will be used to introduce linguistic examples.

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

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

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

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

407        (1.2) The trophy doesn’t fit into the brown suitcase because **it** is too [small/large].  
408        When the final word is *small*, then the pronoun *it* refers to the suitcase; when the final  
409 word is *large*, then *it* refers to the trophy. Solving this example requires spatial reasoning;  
410 other schemas require reasoning about actions and their effects, emotions and intentions,  
411 and social conventions.

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

---

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

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

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

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

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

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

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

## 466 1.2 Three themes in natural language processing

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

### 473 1.2.1 Learning and knowledge

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

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

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

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

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

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

---

<sup>3</sup>The *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).

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

523 (2015).

524 **1.2.2 Search and learning**

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

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

527 where,

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

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

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

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

---

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

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

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

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

### 577    1.2.3 Relational, compositional, and distributional perspectives

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

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

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

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

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

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

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

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

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

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

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

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

### 639 1.3 Learning to do natural language processing

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

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

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

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

#### 655 1.3.1 Background

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

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

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

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

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

### 684 1.3.2 How to use this book

685 The textbook is organized into four main units:

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

---

<sup>6</sup><https://mml-book.github.io/>

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

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

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

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

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

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

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

731 **Acknowledgments**

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



745

## Part I

746

# Learning



747

## Chapter 2

748

# Linear text classification

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

---

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

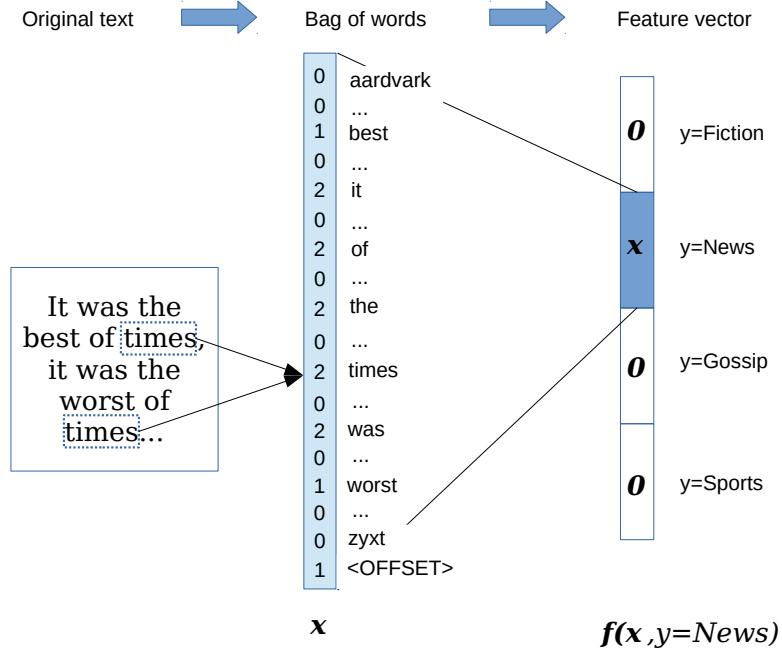


Figure 2.1: The bag-of-words and feature vector representations, for a hypothetical text classification task.

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

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

Returning to the weights  $\theta$ , 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,<sup>2</sup>

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

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  $\prod_{i=1}^N p_{X,Y}(\mathbf{x}^{(i)}, y^{(i)})$ .<sup>3</sup>

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**:

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

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

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

---

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

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

---

**Algorithm 1** Generative process for the Naïve Bayes classifier

---

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

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

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

---

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

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

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

---

<sup>4</sup>Throughout 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.

<sup>5</sup>Generative 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).

<sup>6</sup>Can you think of any cases in which this assumption is too strong?

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

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

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

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

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

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

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

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

### 845 2.1.1 Types and tokens

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

---

<sup>8</sup>Technically, a multinomial distribution requires a second parameter, the total number of word counts in  $x$ . In the bag-of-words representation is equal to the number of words in the document. However, this parameter is irrelevant for classification.

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

**Algorithm 2** Alternative generative process for the Naïve Bayes classifier

---

```

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

```

---

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

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

870 **2.1.2 Prediction**

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

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

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

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

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

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

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

where

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

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

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

### 878 2.1.3 Estimation

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

### 893 2.1.4 Smoothing and MAP estimation

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

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

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

902 This is called **Laplace smoothing**.<sup>10</sup> The pseudocount  $\alpha$  is a **hyperparameter**, because it  
 903 controls the form of the log-likelihood function, which in turn drives the estimation of  $\phi$ .

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

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

### 914 2.1.5 Setting hyperparameters

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

---

<sup>10</sup>Laplace smoothing has a Bayesian justification, in which the generative model is extended to include  $\phi$  as a random variable. The resulting estimate is called **maximum a posteriori**, or MAP.

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

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

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

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

## 947 2.2 Discriminative learning

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

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

- 958     • To better handle out-of-vocabulary terms, we want features that apply to multiple  
 959        words, such as prefixes and suffixes (e.g., *anti-*, *un-*, *-ing*) and capitalization.  
 960     • We also want *n-gram* features that apply to multi-word units: **bigrams** (e.g., *not*  
 961        *good, not bad*), **trigrams** (e.g., *not so bad, lacking any decency, never before imagined*), and  
 962        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:<sup>11</sup>

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

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

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

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

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

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

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

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

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

### 973 2.2.1 Perceptron

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

---

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

**Algorithm 3** Perceptron learning algorithm

---

```

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

```

---

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

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

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

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

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

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

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

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

### 1007 2.2.2 Averaged perceptron

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

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

1025 Even if the data is not separable, the averaged weights will eventually converge. One  
 1026 possible stopping criterion is to check the difference between the average weight vectors  
 1027 after each pass through the data: if the norm of the difference falls below some predefined  
 1028 threshold, we can stop training. Another stopping criterion is to hold out some data,  
 1029 and to measure the predictive accuracy on this heldout data. When the accuracy on the  
 1030 heldout data starts to decrease, the learning algorithm has begun to **overfit** the training

---

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

**Algorithm 4** Averaged perceptron learning algorithm

---

```

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

```

---

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

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

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

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

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

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

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

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

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

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

1044 The problem of minimizing  $\ell_{\text{NB}}$  is thus identical to the problem of maximum-likelihood  
 1045 estimation.

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

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

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

1056 The perceptron optimizes the following loss function:

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

1057 When  $\hat{y} = y^{(i)}$ , the loss is zero; otherwise, it increases linearly with the gap between the  
 1058 score for the predicted label  $\hat{y}$  and the score for the true label  $y^{(i)}$ . Plotting this loss against  
 1059 the input  $\max_{y \in \mathcal{Y}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)})$  gives a hinge shape, motivating the name  
 1060 **hinge loss**.

---

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

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

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

1063 At each instance perceptron algorithm takes a step of magnitude one in the opposite direction  
 1064 of this **gradient**,  $\nabla_{\theta} \ell_{\text{PERCEPTRON}} = \frac{\partial}{\partial \theta} \ell_{\text{PERCEPTRON}}(\theta; \mathbf{x}^{(i)}, y^{(i)})$ . As we will see in § 2.5,  
 1065 this is an example of the optimization algorithm **stochastic gradient descent**, applied to  
 1066 the objective in Equation 2.41.

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

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

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

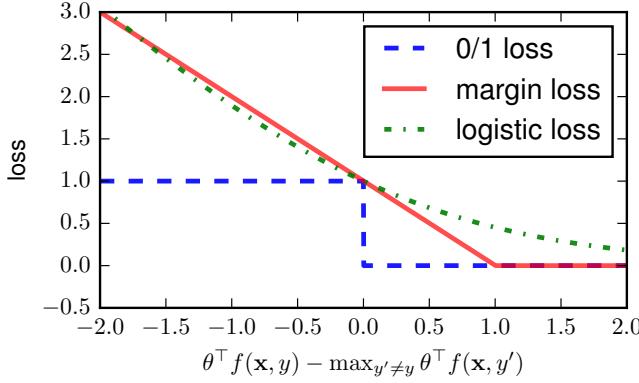


Figure 2.2: Margin, zero-one, and logistic loss functions.

### 1092 2.3.1 Large margin classification

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

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

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

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

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

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

1101 **2.3.2 Support vector machines**

If a dataset is linearly separable, then there is some hyperplane  $\theta$  that correctly classifies all training instances with margin  $\rho$  (by Definition 1). This margin can be increased to any desired value by multiplying the weights by a constant. Now, for any datapoint  $(x^{(i)}, y^{(i)})$ , the geometric distance to the separating hyperplane is given by  $\frac{\gamma(\theta; x^{(i)}, y^{(i)})}{\|\theta\|_2}$ ,

where the denominator is the norm of the weights,  $\|\theta\|_2 = \sqrt{\sum_j \theta_j^2}$ . The geometric distance is sometimes called the **geometric margin**, in contrast to the **functional margin**  $\gamma(\theta; x^{(i)}, y^{(i)})$ . Both are shown in Figure 2.3. The geometric margin is a good measure of the robustness of the separator: if the functional margin is large, but the norm  $\|\theta\|_2$  is also large, then a small change in  $x^{(i)}$  could cause it to be misclassified. We therefore seek to maximize the minimum geometric margin, subject to the constraint that the functional margin is at least one:

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

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

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

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

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



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

### 1112 2.3.3 Slack variables

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

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

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

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

1121 solve for the slack variables,

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

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

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

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

1125 Now define the **cost** of a classification error as,<sup>14</sup>

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

Equation 2.50 can be rewritten using this cost function,

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

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

Differentiating Equation 2.52 with respect to the weights gives,

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

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

1132 where  $L_{\text{SVM}}$  refers to minimization objective in Equation 2.52. This gradient is very similar  
 1133 to the perceptron update. One difference is the additional term  $\lambda \boldsymbol{\theta}$ , which **regularizes** the

---

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

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

## 2.4 Logistic regression

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

**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  $\theta \cdot f(\mathbf{x}, y)$  as a scoring function for the compatibility of the base features  $\mathbf{x}$  and the label  $y$ . To convert this score into a probability, we first exponentiate, obtaining  $\exp(\theta \cdot f(\mathbf{x}, y))$ , which is guaranteed to be non-negative. Next, we normalize, dividing over all possible labels  $y' \in \mathcal{Y}$ . The resulting conditional probability is defined as,

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

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

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

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

The final line is obtained by plugging in Equation 2.55 and taking the logarithm.<sup>15</sup> Inside

---

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

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

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

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

### 1152 2.4.1 Regularization

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

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

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

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

1163 **2.4.2 Gradients**

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

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

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

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

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

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

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

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

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

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

1172 **2.5 Optimization**

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

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

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

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

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

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

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

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

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

### 1192 2.5.1 Batch optimization

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

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

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

---

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

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

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

1196 In practice, gradient descent can be slow to converge, as the gradient can become  
 1197 infinitesimally small. Faster convergence can be obtained by second-order Newton opti-  
 1198 mization, which incorporates the inverse of the **Hessian matrix**,

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

1199 The size of the Hessian matrix is quadratic in the number of features. In the bag-of-words  
 1200 representation, this is usually too big to store, let alone invert. **Quasi-Network optimiza-**  
 1201 **tion** techniques maintain a low-rank approximation to the inverse of the Hessian matrix.  
 1202 Such techniques usually converge more quickly than gradient descent, while remaining  
 1203 computationally tractable even for large feature sets. A popular quasi-Newton algorithm  
 1204 is **L-BFGS** (Liu and Nocedal, 1989), which is implemented in many scientific computing  
 1205 environments, such as `scipy` and `Matlab`.

1206 For any gradient-based technique, the user must set the learning rates  $\eta^{(t)}$ . While con-  
 1207 vergence proofs usually employ a decreasing learning rate, in practice, it is common to fix  
 1208  $\eta^{(t)}$  to a small constant, like  $10^{-3}$ . The specific constant can be chosen by experimentation,  
 1209 although there is research on determining the learning rate automatically (Schaul et al.,  
 1210 2013; Wu et al., 2018).

### 1211 2.5.2 Online optimization

1212 Batch optimization computes the objective on the entire training set before making an up-  
 1213 date. This may be inefficient, because at early stages of training, a small number of train-  
 1214 ing examples could point the learner in the correct direction. **Online learning** algorithms  
 1215 make updates to the weights while iterating through the training data. The theoretical  
 1216 basis for this approach is a stochastic approximation to the true objective function,

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

1217 where the instance  $(\mathbf{x}^{(j)}, y^{(j)})$  is sampled at random from the full dataset.

1218 In **stochastic gradient descent**, the approximate gradient is computed by randomly  
 1219 sampling a single instance, and an update is made immediately. This is similar to the  
 1220 perceptron algorithm, which also updates the weights one instance at a time. In **mini-**  
 1221 **batch** stochastic gradient descent, the gradient is computed over a small set of instances.  
 1222 A typical approach is to set the minibatch size so that the entire batch fits in memory on a  
 1223 graphics processing unit (GPU; Neubig et al., 2017). It is then possible to speed up learn-  
 1224 ing by parallelizing the computation of the gradient over each instance in the minibatch.

---

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

---

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

---

```

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

```

---

1225     Algorithm 5 offers a generalized view of gradient descent. In standard gradient de-  
 1226     scendent, the batcher returns a single batch with all the instances. In stochastic gradient de-  
 1227     scent, it returns  $N$  batches with one instance each. In mini-batch settings, the batcher  
 1228     returns  $B$  minibatches,  $1 < B < N$ .

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

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

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

1229 where  $j$  iterates over features in  $\mathbf{f}(\mathbf{x}, y)$ .

1230     In most cases, the number of active features for any instance is much smaller than the  
 1231     number of weights. If so, the computation cost of online optimization will be dominated  
 1232     by the update from the regularization term,  $\lambda \boldsymbol{\theta}$ . The solution is to be “lazy”, updating  
 1233     each  $\theta_j$  only as it is used. To implement lazy updating, store an additional parameter  $\tau_j$ ,  
 1234     which is the iteration at which  $\theta_j$  was last updated. If  $\theta_j$  is needed at time  $t$ , the  $t - \tau$

1235 regularization updates can be performed all at once. This strategy is described in detail  
 1236 by Kummerfeld et al. (2015).

## 1237 2.6 \*Additional topics in classification

1238 Throughout this text, advanced topics will be marked with an asterisk.

### 1239 2.6.1 Feature selection by regularization

1240 In logistic regression and large-margin classification, generalization can be improved by  
 1241 regularizing the weights towards 0, using the  $L_2$  norm. But rather than encouraging  
 1242 weights to be small, it might be better for the model to be **sparse**: it should assign weights  
 1243 of exactly zero to most features, and only assign non-zero weights to features that are  
 1244 clearly necessary. This idea can be formalized by the  $L_0$  norm,  $L_0 = \|\theta\|_0 = \sum_j \delta(\theta_j \neq 0)$ ,  
 1245 which applies a constant penalty for each non-zero weight. This norm can be thought  
 1246 of as a form of **feature selection**: optimizing the  $L_0$ -regularized conditional likelihood is  
 1247 equivalent to trading off the log-likelihood against the number of active features. Reduc-  
 1248 ing the number of active features is desirable because the resulting model will be fast,  
 1249 low-memory, and should generalize well, since irrelevant features will be pruned away.  
 1250 Unfortunately, the  $L_0$  norm is non-convex and non-differentiable. Optimization under  $L_0$   
 1251 regularization is **NP-hard**, meaning that it can be solved efficiently only if P=NP (Ge et al.,  
 1252 2011).

1253 A useful alternative is the  $L_1$  norm, which is equal to the sum of the absolute values  
 1254 of the weights,  $\|\theta\|_1 = \sum_j |\theta_j|$ . The  $L_1$  norm is convex, and can be used as an approxima-  
 1255 tion to  $L_0$  (Tibshirani, 1996). Conveniently, the  $L_1$  norm also performs feature selection,  
 1256 by driving many of the coefficients to zero; it is therefore known as a **sparsity inducing**  
 1257 **regularizer**. The  $L_1$  norm does not have a gradient at  $\theta_j = 0$ , so we must instead optimize  
 1258 the  $L_1$ -regularized objective using **subgradient** methods. The associated stochastic sub-  
 1259 gradient descent algorithms are only somewhat more complex than conventional SGD;  
 1260 Sra et al. (2012) survey approaches for estimation under  $L_1$  and other regularizers.

1261 Gao et al. (2007) compare  $L_1$  and  $L_2$  regularization on a suite of NLP problems, finding  
 1262 that  $L_1$  regularization generally gives similar accuracy to  $L_2$  regularization, but that  $L_1$   
 1263 regularization produces models that are between ten and fifty times smaller, because more  
 1264 than 90% of the feature weights are set to zero.

### 1265 2.6.2 Other views of logistic regression

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

inner product through a **logistic function**,

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

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

1266 This is the origin of the name **logistic regression**. Logistic regression can be viewed as  
 1267 part of a larger family of **generalized linear models** (GLMs), in which various other “link  
 1268 functions” convert between the inner product  $\boldsymbol{\theta} \cdot \mathbf{x}$  and the parameter of a conditional  
 1269 probability distribution.

1270 In the early NLP literature, logistic regression is frequently called **maximum entropy**  
 1271 classification (Berger et al., 1996). This name refers to an alternative formulation, in  
 1272 which the goal is to find the maximum entropy probability function that satisfies **moment-**  
 1273 **matching** constraints. These constraints specify that the empirical counts of each feature  
 1274 should match the expected counts under the induced probability distribution  $p_{Y|X;\boldsymbol{\theta}}$ .

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

1275 The moment-matching constraint is satisfied exactly when the derivative of the condi-  
 1276 tional log-likelihood function (Equation 2.64) is equal to zero. However, the constraint  
 1277 can be met by many values of  $\boldsymbol{\theta}$ , so which should we choose?

1278 The **entropy** of the conditional probability distribution  $p_{Y|X}$  is,

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

1279 where  $\mathcal{X}$  is the set of all possible feature vectors, and  $p_X(\mathbf{x})$  is the probability of observing  
 1280 the base features  $\mathbf{x}$ . The distribution  $p_X$  is unknown, but it can be estimated by summing  
 1281 over all the instances in the training set,

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

1282 If the entropy is large, the likelihood function is smooth across possible values of  $y$ ;  
 1283 if it is small, the likelihood function is sharply peaked at some preferred value; in the  
 1284 limiting case, the entropy is zero if  $p(y | x) = 1$  for some  $y$ . The maximum-entropy cri-  
 1285 terion chooses to make the weakest commitments possible, while satisfying the moment-  
 1286 matching constraints from Equation 2.78. The solution to this constrained optimization  
 1287 problem is identical to the maximum conditional likelihood (logistic-loss) formulation  
 1288 that was presented in § 2.4.

## 1289 2.7 Summary of learning algorithms

1290 It is natural to ask which learning algorithm is best, but the answer depends on what  
 1291 characteristics are important to the problem you are trying to solve.

1292 **Naïve Bayes** *Pros:* easy to implement; estimation is fast, requiring only a single pass over  
 1293 the data; assigns probabilities to predicted labels; controls overfitting with smoothing  
 1294 parameter. *Cons:* often has poor accuracy, especially with correlated features.

1295 **Perceptron** *Pros:* easy to implement; online; error-driven learning means that accuracy  
 1296 is typically high, especially after averaging. *Cons:* not probabilistic; hard to know  
 1297 when to stop learning; lack of margin can lead to overfitting.

1298 **Support vector machine** *Pros:* optimizes an error-based metric, usually resulting in high  
 1299 accuracy; overfitting is controlled by a regularization parameter. *Cons:* not proba-  
 1300 bilistic.

1301 **Logistic regression** *Pros:* error-driven and probabilistic; overfitting is controlled by a reg-  
 1302 ularization parameter. *Cons:* batch learning requires black-box optimization; logistic  
 1303 loss can “overtrain” on correctly labeled examples.

1304 One of the main distinctions is whether the learning algorithm offers a probability  
 1305 over labels. This is useful in modular architectures, where the output of one classifier  
 1306 is the input for some other system. In cases where probability is not necessary, the sup-  
 1307 port vector machine is usually the right choice, since it is no more difficult to implement  
 1308 than the perceptron, and is often more accurate. When probability is necessary, logistic  
 1309 regression is usually more accurate than Naïve Bayes.

## 1310 Additional resources

1311 For more on classification, you can consult a textbook on machine learning (e.g., Mur-  
 1312 phy, 2012), although the notation will differ slightly from what is typical in natural lan-  
 1313 guage processing. Probabilistic methods are surveyed by Hastie et al. (2009), and Mohri  
 1314 et al. (2012) emphasize theoretical considerations. Online learning is a rapidly moving  
 1315 subfield of machine learning, and Bottou et al. (2016) describes progress through 2016.  
 1316 Kummerfeld et al. (2015) empirically review several optimization algorithms for large-  
 1317 margin learning. The python toolkit `scikit-learn` includes implementations of all of  
 1318 the algorithms described in this chapter (Pedregosa et al., 2011).

1319 **Exercises**

- 1320 1. Let  $\mathbf{x}$  be a bag-of-words vector such that  $\sum_{j=1}^V x_j = 1$ . Verify that the multinomial  
1321 probability  $p_{\text{mult}}(\mathbf{x}; \phi)$ , as defined in Equation 2.12, is identical to the probability of  
1322 the same document under a categorical distribution,  $p_{\text{cat}}(\mathbf{w}; \phi)$ .
- 1323 2. Derive the maximum-likelihood estimate for the parameter  $\mu$  in Naïve Bayes.
- 1324 3. As noted in the discussion of averaged perceptron in § 2.2.2, the computation of the  
1325 running sum  $\mathbf{m} \leftarrow \mathbf{m} + \boldsymbol{\theta}$  is unnecessarily expensive, requiring  $K \times V$  operations.  
1326 Give an alternative way to compute the averaged weights  $\bar{\boldsymbol{\theta}}$ , with complexity that is  
1327 independent of  $V$  and linear in the sum of feature sizes  $\sum_{i=1}^N |\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)})|$ .
- 1328 4. Consider a dataset that is comprised of two identical instances  $\mathbf{x}^{(1)} = \mathbf{x}^{(2)}$  with  
1329 distinct labels  $y^{(1)} \neq y^{(2)}$ . Assume all features are binary  $x_j \in \{0, 1\}$  for all  $j$ .

1330 Now suppose that the averaged perceptron always chooses  $i = 1$  when  $t$  is even,  
1331 and  $i = 2$  when  $t$  is odd, and that it will terminate under the following condition:

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

1332 In words, the algorithm stops when the largest change in the averaged weights is  
1333 less than or equal to  $\epsilon$ . Compute the number of iterations before the averaged per-  
1334 ceptron terminates.

- 1335 5. Suppose you have two labeled datasets  $D_1$  and  $D_2$ , with the same features and la-  
1336 bels.
  - 1337 • Let  $\boldsymbol{\theta}^{(1)}$  be the unregularized logistic regression (LR) coefficients from training  
1338 on dataset  $D_1$ .
  - 1339 • Let  $\boldsymbol{\theta}^{(2)}$  be the unregularized LR coefficients (same model) from training on  
1340 dataset  $D_2$ .
  - 1341 • Let  $\boldsymbol{\theta}^*$  be the unregularized LR coefficients from training on the combined  
1342 dataset  $D_1 \cup D_2$ .

Under these conditions, prove that for any feature  $j$ ,

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

1343



1344 

## Chapter 3

1345 

# Nonlinear classification

1346 Linear classification may seem like all we need for natural language processing. The bag-  
1347 of-words representation is inherently high dimensional, and the number of features is  
1348 often larger than the number of training instances. This means that it is usually possible  
1349 to find a linear classifier that perfectly fits the training data. Moving to nonlinear classifi-  
1350 cation may therefore only increase the risk of overfitting. For many tasks, **lexical features**  
1351 (words) are meaningful in isolation, and can offer independent evidence about the in-  
1352 stance label — unlike computer vision, where individual pixels are rarely informative,  
1353 and must be evaluated holistically to make sense of an image. For these reasons, natu-  
1354 ral language processing has historically focused on linear classification to a greater extent  
1355 than other machine learning application domains.

1356 But in recent years, nonlinear classifiers have swept through natural language pro-  
1357 cessing, and are now the default approach for many tasks (Manning, 2016). There are at  
1358 least three reasons for this change.

- 1359 • There have been rapid advances in **deep learning**, a family of nonlinear meth-  
1360 ods that learn complex functions of the input through multiple layers of computa-  
1361 tion (Goodfellow et al., 2016).
- 1362 • Deep learning facilitates the incorporation of **word embeddings**, which are dense  
1363 vector representations of words. Word embeddings can be learned from large amounts  
1364 of unlabeled data, and enable generalization to words that do not appear in the an-  
1365notated training data (word embeddings are discussed in detail in chapter 14).
- 1366 • A third reason for the rise of deep nonlinear learning algorithms is hardware. Many  
1367 deep learning models can be implemented efficiently on graphics processing units  
1368 (GPUs), offering substantial performance improvements over CPU-based comput-  
1369 ing.

1370 This chapter focuses on **neural networks**, which are the dominant approach for non-

1371 linear classification in natural language processing today.<sup>1</sup> Historically, a few other non-  
 1372 linear learning methods have been applied to language data:

- 1373     • **Kernel methods** are generalizations of the **nearest-neighbor** classification rule, which  
 1374       classifies each instance by the label of the most similar example in the training  
 1375       set (Hastie et al., 2009). The application of the **kernel support vector machine** to  
 1376       information extraction is described in chapter 17.
- 1377     • **Decision trees** classify instances by checking a set of conditions. Scaling decision  
 1378       trees to bag-of-words inputs is difficult, but decision trees have been successful in  
 1379       problems such as coreference resolution (chapter 15), where more compact feature  
 1380       sets can be constructed (Soon et al., 2001).
- 1381     • **Boosting** and related **ensemble methods** work by combining the predictions of sev-  
 1382       eral “weak” classifiers, each of which may consider only a small subset of features.  
 1383       Boosting has been successfully applied to text classification (Schapire and Singer,  
 1384       2000) and syntactic analysis (Abney et al., 1999), and remains one of the most suc-  
 1385       cessful methods on machine learning competition sites such as Kaggle (Chen and  
 1386       Guestrin, 2016).

### 1387 3.1 Feedforward neural networks

1388 Consider the problem of building a classifier for movie reviews. The goal is to predict  
 1389 a label  $y \in \{\text{GOOD}, \text{BAD}, \text{OKAY}\}$  from a representation of the text of each document,  $x$ .  
 1390 But what makes a good movie? The story, acting, cinematography, soundtrack, and so  
 1391 on. Now suppose the training set contains labels for each of these additional features,  
 1392  $z = [z_1, z_2, \dots, z_{K_z}]^\top$ . With such information, we could build a two-step classifier:

- 1393     1. **Use the text  $x$  to predict the features  $z$ .** Specifically, train a logistic regression clas-  
 1394       sifier to compute  $p(z_k | x)$ , for each  $k \in \{1, 2, \dots, K_z\}$ .
- 1395     2. **Use the features  $z$  to predict the label  $y$ .** Again, train a logistic regression classifier  
 1396       to compute  $p(y | z)$ . On test data,  $z$  is unknown, so we use the probabilities  $p(z | x)$   
 1397       from the first layer as the features.

1398 This setup is shown in Figure 3.1, which describes the proposed classifier in a **compu-  
 1399 tation graph**: the text features  $x$  are connected to the middle layer  $z$ , which in turn is  
 1400 connected to the label  $y$ .

1401 Since each  $z_k \in \{0, 1\}$ , we can treat  $p(z_k | x)$  as a binary classification problem, using  
 1402 binary logistic regression:

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

---

<sup>1</sup>I will use “deep learning” and “neural networks” interchangeably.

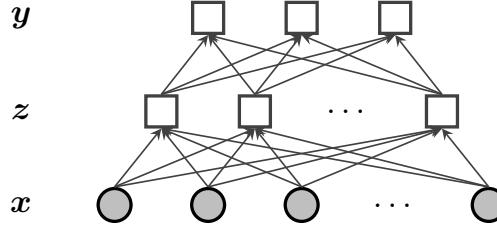


Figure 3.1: A feedforward neural network. Shaded circles indicate observed features, usually words; squares indicate nodes in the computation graph, which are computed from the information carried over the incoming arrows.

1403 where  $\sigma(\cdot)$  is the **sigmoid** function (shown in Figure 3.2), and the matrix  $\Theta^{(x \rightarrow z)} \in \mathbb{R}^{K_z \times V}$   
 1404 is constructed by stacking the weight vectors for each  $z_k$ ,

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

1405 We will assume that  $x$  contains a term with a constant value of 1, so that a corresponding  
 1406 offset parameter is included in each  $\theta_k^{(x \rightarrow z)}$ .

1407 The output layer is computed by the multi-class logistic regression probability,

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

1408 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  
 1409 constructed by concatenation,

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

1410 The vector of probabilities over each possible value of  $y$  is denoted,

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

1411 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,

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

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

1412 where  $\sigma(\cdot)$  is now applied **elementwise** to the vector of inner products,

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

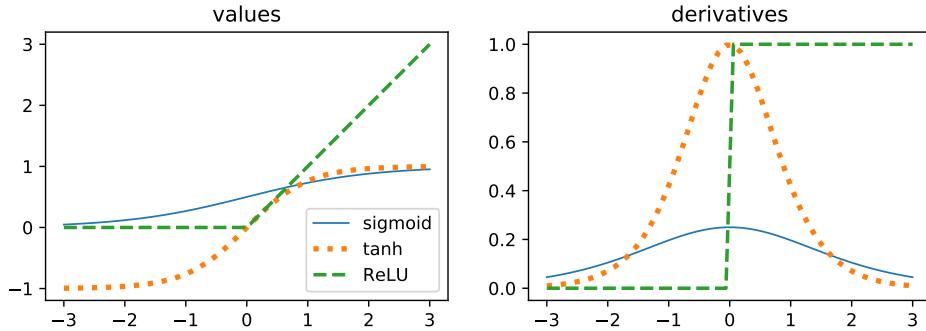


Figure 3.2: The sigmoid, tanh, and ReLU activation functions

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

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

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

1413 This defines a classification model that predicts the label  $y \in \mathcal{Y}$  from the base features  $x$ ,  
1414 through a “hidden layer”  $z$ . This is a **feedforward neural network**.<sup>2</sup>

## 1415 3.2 Designing neural networks

1416 This feedforward neural network can be generalized in a number of ways.

### 1417 3.2.1 Activation functions

1418 If the hidden layer is viewed as a set of latent features, then the sigmoid function repre-  
1419 presents the extent to which each of these features is “activated” by a given input. However,  
1420 the hidden layer can be regarded more generally as a nonlinear transformation of the in-  
1421 put. This opens the door to many other activation functions, some of which are shown in  
1422 Figure 3.2. At the moment, the choice of activation functions is more art than science, but  
1423 a few points can be made about the most popular varieties:

- 1424 • The range of the sigmoid function is  $(0, 1)$ . The bounded range ensures that a cas-  
1425 cade of sigmoid functions will not “blow up” to a huge output, and this is impor-

---

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

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

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

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

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

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

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

### 3.2.2 Network structure

Deep networks stack up several hidden layers, with each  $z^{(d)}$  acting as the input to the next layer,  $z^{(d+1)}$ . As the total number of nodes in the network increases, so does its capacity to learn complex functions of the input. 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.<sup>3</sup>

---

<sup>3</sup>With 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

1453 It is also possible to “short circuit” a hidden layer, by propagating information directly  
 1454 from the input to the next higher level of the network. This is the idea behind **residual net-**  
 1455 **works**, which propagate information directly from the input to the subsequent layer (He  
 1456 et al., 2016),

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

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

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

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

1457 where  $\odot$  refers to an elementwise vector product, and  $\mathbf{1}$  is a column vector of ones. The  
 1458 sigmoid function is applied elementwise to its input; recall that the output of this function  
 1459 is restricted to the range  $[0, 1]$ . Gating is also used in the **long short-term memory (LSTM)**,  
 1460 which is discussed in chapter 6. Residual and highway connections address a problem  
 1461 with deep architectures: repeated application of a nonlinear activation function can make  
 1462 it difficult to learn the parameters of the lower levels of the network, which are too distant  
 1463 from the supervision signal.

### 1464 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**,

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

1465 where  $\Theta = \{\Theta^{(x \rightarrow z)}, \Theta^{(z \rightarrow y)}, \mathbf{b}\}$  is the entire set of parameters.

This loss can be written alternatively as follows:

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

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

1466 where  $e_{y^{(i)}}$  is a **one-hot vector** of zeros with a value of 1 at position  $y^{(i)}$ . The inner product  
 1467 between  $e_{y^{(i)}}$  and  $\log \tilde{y}$  is also called the multinomial **cross-entropy**, and this terminology  
 1468 is preferred in many neural networks papers and software packages.

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

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

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

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

- 1469 In regression problems, the output is a scalar or vector (see § 4.1.2). For these problems, a  
 1470 typical loss function is the squared error  $(y - \hat{y})^2$  or squared norm  $\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2$ .

### 1471 3.2.4 Inputs and lookup layers

1472 In text classification, the input layer  $\mathbf{x}$  can refer to a bag-of-words vector, where  $x_j$  is  
 1473 the count of word  $j$ . The input to the hidden unit  $z_k$  is then  $\sum_{j=1}^V \theta_{j,k}^{(x \rightarrow z)} x_j$ , and word  $j$  is  
 1474 represented by the vector  $\theta_j^{(x \rightarrow z)}$ . This vector is sometimes described as the **embedding** of  
 1475 word  $j$ , and can be learned from unlabeled data, using techniques discussed in chapter 14.  
 1476 The columns of  $\Theta^{(x \rightarrow z)}$  are each  $K_z$ -dimensional word embeddings.

1477 Chapter 2 presented an alternative view of text documents, as a sequence of word  
 1478 tokens,  $w_1, w_2, \dots, w_M$ . In a neural network, each word token  $w_m$  is represented with  
 1479 a one-hot vector,  $e_{w_m} \in \mathbb{R}^V$ . The matrix-vector product  $\Theta^{(x \rightarrow z)} e_{w_m}$  returns the embed-  
 1480 ding of word  $w_m$ . The complete document can be represented by horizontally concatenating  
 1481 these one-hot vectors,  $\mathbf{W} = [e_{w_1}, e_{w_2}, \dots, e_{w_M}]$ , and the bag-of-words representation can  
 1482 be recovered from the matrix-vector product  $\mathbf{W} \mathbf{1}$ , which simply sums each row over the  
 1483 tokens  $m = \{1, 2, \dots, M\}$ . The matrix product  $\Theta^{(x \rightarrow z)} \mathbf{W}$  contains the horizontally con-  
 1484 catenated embeddings of each word in the document, which will be useful as the starting  
 1485 point for **convolutional neural networks** (see § 3.4). This is sometimes called a **lookup**  
 1486 **layer**, because the first step is to lookup the embeddings for each word in the input text.

## 1487 3.3 Learning neural networks

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

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

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

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

- 1488 where  $f$  is an elementwise activation function, such as  $\sigma$  or ReLU.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1493 For intuition, consider each of the terms in the product.

- 1494 • 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  
1495 doesn't matter how  $z_k$  is computed, and so  $\frac{\partial \ell^{(i)}}{\partial \theta_{n,k}^{(x \rightarrow z)}} \rightarrow 0$ .
- 1496 • If  $z_k$  is near 1 or 0, then the curve of the sigmoid function (Figure 3.2) is nearly flat,  
1497 and changing the inputs will make little local difference. The term  $z_k \times (1 - z_k)$  is  
1498 maximized at  $z_k = \frac{1}{2}$ , where the slope of the sigmoid function is steepest.
- 1499 • If  $x_n = 0$ , then it does not matter how we set the weights  $\theta_{n,k}^{(x \rightarrow z)}$ , so  $\frac{\partial \ell^{(i)}}{\partial \theta_{n,k}^{(x \rightarrow z)}} = 0$ .

### 1500 3.3.1 Backpropagation

1501 In the equations above, the value  $\frac{\partial \ell^{(i)}}{\partial z_k}$  is reused in the derivatives with respect to each  
1502  $\theta_{n,k}^{(x \rightarrow z)}$ . It should therefore be computed once, and then cached. Furthermore, we should  
1503 only compute any derivative once we have already computed all of the necessary "inputs"  
1504 demanded by the chain rule of differentiation. This combination of sequencing, caching,  
1505 and differentiation is known as **backpropagation**. It can be generalized to any directed  
1506 acyclic **computation graph**.

1507 A computation graph is a declarative representation of a computational process. At  
1508 each node  $t$ , compute a value  $v_t$  by applying a function  $f_t$  to a (possibly empty) list of  
1509 parent nodes,  $\pi_t$ . For example, in a feedforward network with one hidden layer, there are  
1510 nodes for the input  $\mathbf{x}^{(i)}$ , the hidden layer  $\mathbf{z}$ , the predicted output  $\tilde{\mathbf{y}}$ , and the parameters  
1511  $\{\Theta^{(x \rightarrow z)}, \Theta^{(z \rightarrow y)}, \mathbf{b}\}$ . During training, there is also a node for the observed label  $y^{(i)}$  and  
1512 the loss  $\ell^{(i)}$ . Computation graphs have three main types of nodes:

1513 **Variables.** The variables include the *inputs*  $\mathbf{x}$ , the *hidden nodes*  $\mathbf{z}$ , the outputs  $\mathbf{y}$ , and the  
1514 loss function. Inputs are variables that do not have parents. Backpropagation com-  
1515putes the gradients with respect to all variables except the inputs, but does not up-  
1516 date the variables during learning.

1517 **Parameters.** In a feedforward network, the parameters include the weights and offsets.  
1518 Parameter nodes do not have parents, and they are updated during learning.

---

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

---

```

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

```

---

1519   **Objective.** The *objective* node is not the parent of any other node. Backpropagation begins  
 1520   by computing the gradient with respect to this node.

1521   If the computation graph is a directed acyclic graph, then it is possible to order the  
 1522   nodes with a topological sort, so that if node  $t$  is a parent of node  $t'$ , then  $t < t'$ . This  
 1523   means that the values  $\{v_t\}_{t=1}^T$  can be computed in a single forward pass. The topolog-  
 1524   ical sort is reversed when computing gradients: each gradient  $g_t$  is computed from the  
 1525   gradients of the children of  $t$ , implementing the chain rule of differentiation. The general  
 1526   backpropagation algorithm for computation graphs is shown in Algorithm 6, and illus-  
 1527   trated in Figure 3.3.

1528   While the gradients with respect to each parameter may be complex, they are com-  
 1529   posed of products of simple parts. For many networks, all gradients can be computed  
 1530   through **automatic differentiation**. This means that end users need only specify the feed-  
 1531   forward computation, and the gradients necessary for learning can be obtained automati-  
 1532   cally. There are many software libraries that perform automatic differentiation on compu-  
 1533   tation graphs, such as Torch (Collobert et al., 2011), TensorFlow (Abadi et al., 2016), and  
 1534   DyNet (Neubig et al., 2017). One important distinction between these libraries is whether  
 1535   they support **dynamic computation graphs**, in which the structure of the computation  
 1536   graph varies across instances. Static computation graphs are compiled in advance, and  
 1537   can be applied to fixed-dimensional data, such as bag-of-words vectors. In many natu-  
 1538   ral language processing problems, each input has a distinct structure, requiring a unique  
 1539   computation graph.

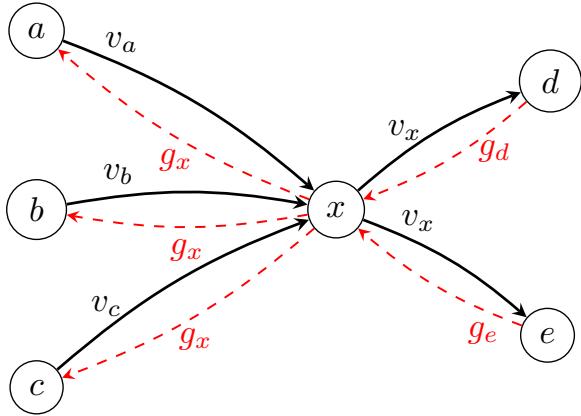


Figure 3.3: Backpropagation at a single node  $x$  in the computation graph. The values of the predecessors  $v_a, v_b, v_c$  are the inputs to  $x$ , which computes  $v_x$ , and passes it on to the successors  $d$  and  $e$ . The gradients at the successors  $g_d$  and  $g_e$  are passed back to  $x$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

### 1540 3.3.2 Regularization and dropout

1541 In linear classification, overfitting was addressed by augmenting the objective with a reg-  
 1542 ularization term,  $\lambda \|\theta\|_2^2$ . This same approach can be applied to feedforward neural net-  
 1543 works, penalizing each matrix of weights:

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

1544 where  $\|\Theta\|_F^2 = \sum_{i,j} \theta_{i,j}^2$  is the squared **Frobenius norm**, which generalizes the  $L_2$  norm  
 1545 to matrices. The bias parameters  $b$  are not regularized, as they do not contribute to the  
 1546 sensitivity of the classifier to the inputs. In gradient-based optimization, the practical  
 1547 effect of Frobenius norm regularization is that the weights “decay” towards zero at each  
 1548 update, motivating the alternative name **weight decay**.

1549 Another approach to controlling model complexity is **dropout**, which involves ran-  
 1550 domly setting some computation nodes to zero during training (Srivastava et al., 2014).  
 1551 For example, in the feedforward network, on each training instance, with probability  $\rho$  we  
 1552 set each input  $x_n$  and each hidden layer node  $z_k$  to zero. Srivastava et al. (2014) recom-  
 1553 mend  $\rho = 0.5$  for hidden units, and  $\rho = 0.2$  for input units. Dropout is also incorporated  
 1554 in the gradient computation, so if node  $z_k$  is dropped, then none of the weights  $\theta_k^{(x \rightarrow z)}$  will  
 1555 be updated for this instance. Dropout prevents the network from learning to depend too  
 1556 much on any one feature or hidden node, and prevents **feature co-adaptation**, in which a

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

### 3.3.3 \*Learning theory

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

The non-convexity of hidden layer neural networks can also be seen by permuting the elements of the hidden layer, from  $z = [z_1, z_2, \dots, z_{K_z}]$  to  $\tilde{z} = [z_{\pi(1)}, z_{\pi(2)}, \dots, z_{\pi(K_z)}]$ . This corresponds to applying  $\pi$  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  $\Theta$  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  $x = (0, 0)$ .<sup>4</sup> 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

---

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

1593 stochastic gradient descent, and by feature noising techniques such as dropout, can help  
 1594 online optimization to escape saddle points and find high-quality optima (Ge et al., 2015).  
 1595 Other techniques address saddle points directly, using local reconstructions of the Hessian  
 1596 matrix (Dauphin et al., 2014) or higher-order derivatives (Anandkumar and Ge, 2016).

1597 **3.3.4 Tricks**

1598 Getting neural networks to work effectively sometimes requires heuristic “tricks” (Bottou,  
 1599 2012; Goodfellow et al., 2016; Goldberg, 2017b). This section presents some tricks that are  
 1600 especially important.

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

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

[3.37]

1601 For the weights leading to a ReLU activation function, He et al. (2015) use similar argu-  
 1602 mentation to justify sampling from a zero-mean Gaussian distribution,

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

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

values sampled from a standard normal distribution:

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

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

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

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

1603 The matrix  $\mathbf{U}$  contains the **singular vectors** of  $\mathbf{A}$ , and is guaranteed to be orthonormal.  
 1604 For more on singular value decomposition, see chapter 14.

1605 Even with careful initialization, there can still be significant variance in the final re-  
 1606 sults. It can be useful to make multiple training runs, and select the one with the best  
 1607 performance on a heldout development set.

1608 **Clipping and normalizing the gradients** As already discussed, the magnitude of the  
 1609 gradient can pose problems for learning: too large, and learning can diverge, with suc-  
 1610 cessive updates thrashing between increasingly extreme values; too small, and learning can  
 1611 grind to a halt. Several heuristics have been proposed to address this issue.

1612 • In **gradient clipping** (Pascanu et al., 2013), an upper limit is placed on the norm of  
 1613 the gradient, and the gradient is rescaled when this limit is exceeded,

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

1614 • In **batch normalization** (Ioffe and Szegedy, 2015), the inputs to each computation  
 1615 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:

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

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

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

1614 Empirically, this speeds convergence of deep architectures. One explanation is that  
 1615 it helps to correct for changes in the distribution of activations during training.

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

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

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

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

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

1616 Layer normalization has similar motivations to batch normalization, but it can be  
 1617 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,

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

1618 where  $g_{t,j}$  is the gradient with respect to parameter  $j$  at time  $t$ , and  $\beta \in [0, 1]$ . This term  
 1619 places more emphasis on recent gradients, and is employed in the **AdaDelta** (Zeiler, 2012)  
 1620 and **Adam** (Kingma and Ba, 2014) optimizers. Online optimization and its theoretical  
 1621 background are reviewed by Bottou et al. (2016). **Early stopping**, mentioned in § 2.2.2,  
 1622 can help to avoid overfitting, by terminating training after reaching a plateau in the per-  
 1623 formance on a heldout validation set.

## 1624 3.4 Convolutional neural networks

1625 A basic weakness of the bag-of-words model is its inability to account for the ways in  
 1626 which words combine to create meaning, including even simple reversals such as *not*  
 1627 *pleasant, hardly a generous offer*, and *I wouldn't mind missing the flight*. Similarly, computer  
 1628 vision faces the challenge of identifying the semantics of images from pixel features that  
 1629 are uninformative in isolation. An earlier generation of computer vision research fo-  
 1630 cused on designing *filters* to aggregate local pixel-level features into more meaningful  
 1631 representations, such as edges and corners (e.g., Canny, 1987). Similarly, earlier NLP re-  
 1632 search attempted to capture multiword linguistic phenomena by hand-designed lexical  
 1633 patterns (Hobbs et al., 1997). In both cases, the output of the filters and patterns could

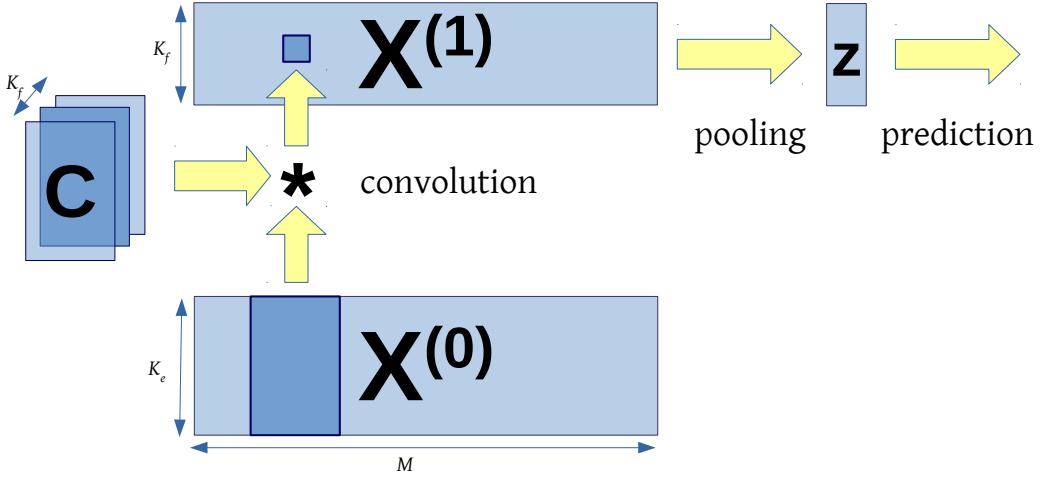


Figure 3.4: A convolutional neural network for text classification

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

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

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

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

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

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

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

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

1642 thereby representing trigram units like *not so unpleasant*. In **one-dimensional convolution**,  
 1643 each filter matrix  $\mathbf{C}^{(k)}$  is constrained to have non-zero values only at row  $k$  (Kalchbrenner et al., 2014).

1645 To deal with the beginning and end of the input, the base matrix  $\mathbf{X}^{(0)}$  may be padded  
 1646 with  $h$  column vectors of zeros at the beginning and end; this is known as **wide convolution**. If padding is not applied, then the output from each layer will be  $h - 1$  units smaller  
 1647 than the input; this is known as **narrow convolution**. The filter matrices need not have  
 1648 identical filter widths, so more generally we could write  $h_k$  to indicate width of filter  
 1649  $\mathbf{C}^{(k)}$ . As suggested by the notation  $\mathbf{X}^{(0)}$ , multiple layers of convolution may be applied,  
 1650 so that  $\mathbf{X}^{(d)}$  is the input to  $\mathbf{X}^{(d+1)}$ .

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

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

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

1652 The vector  $\mathbf{z}$  can now act as a layer in a feedforward network, culminating in a prediction  
 1653  $\hat{y}$  and a loss  $\ell^{(i)}$ . The setup is shown in Figure 3.4.

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

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

1654 The computer vision literature has produced a huge variety of convolutional architectures,  
 1655 and many of these bells and whistles can be applied to text data. One avenue for  
 1656 improvement is more complex pooling operations, such as  $k$ -max pooling (Kalchbrenner  
 1657 et al., 2014), which returns a matrix of the  $k$  largest values for each filter. Another innovation  
 1658 is the use of **dilated convolution** to build multiscale representations (Yu and Koltun,  
 1659 2016). At each layer, the convolutional operator applied in *strides*, skipping ahead by  $s$   
 1660 steps after each feature. As we move up the hierarchy, each layer is  $s$  times smaller than  
 1661 the layer below it, effectively summarizing the input. This idea is shown in Figure 3.5.  
 1662 Multi-layer convolutional networks can also be augmented with “shortcut” connections,  
 1663 as in the ResNet model from § 3.2.2 (Johnson and Zhang, 2017).

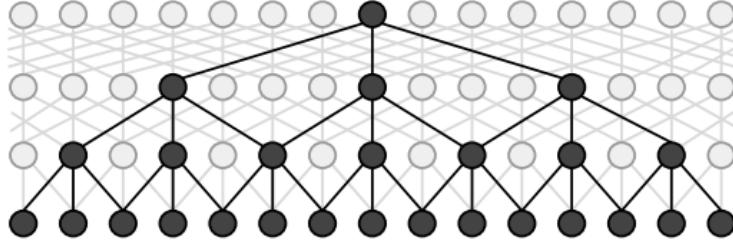


Figure 3.5: A dilated convolutional neural network captures progressively larger context through recursive application of the convolutional operator (Strubell et al., 2017) [todo: permission]

## 1664 Additional resources

1665 The deep learning textbook by Goodfellow et al. (2016) covers many of the topics in this  
 1666 chapter in more detail. For a comprehensive review of neural networks in natural lan-  
 1667 guage processing, see (Goldberg, 2017b). A seminal work on deep learning in natural  
 1668 language processing is the aggressively titled “Natural Language Processing (Almost)  
 1669 from Scratch”, which uses convolutional neural networks to perform a range of language  
 1670 processing tasks (Collobert et al., 2011). This chapter focuses on feedforward and con-  
 1671 volutional neural networks, but recurrent neural networks are one of the most important  
 1672 deep learning architectures for natural language processing. They are covered extensively  
 1673 in chapters 6 and 7.

1674 The role of deep learning in natural language processing research has caused angst  
 1675 in some parts of the natural language processing research community (e.g., Goldberg,  
 1676 2017a), especially as some of the more zealous deep learning advocates have argued that  
 1677 end-to-end learning from “raw” text can eliminate the need for linguistic constructs such  
 1678 as sentences, phrases, and even words (Zhang et al., 2015, originally titled *Text understand-  
 1679 ing from scratch*). These developments were surveyed by Manning (2016).

## 1680 Exercises

- 1681 1. Prove that the softmax and sigmoid functions are equivalent when the number of  
 1682 possible labels is two. Specifically, for any  $\Theta^{(z \rightarrow y)}$  (omitting the offset  $b$  for sim-  
 1683 plicity), show how to construct a vector of weights  $\theta$  such that,

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

- 1684 2. Design a feedforward network to compute the XOR function:

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

1685 Your network should have a single output node which uses the Sign activation func-  
 1686 tion. Use a single hidden layer, with ReLU activation functions. Describe all weights  
 1687 and offsets.

- 1688 3. Consider the same network as above (with ReLU activations for the hidden layer),  
 1689 with an arbitrary differentiable loss function  $\ell(y^{(i)}, \tilde{y})$ , where  $\tilde{y}$  is the activation of  
 1690 the output node. Suppose all weights and offsets are initialized to zero. Prove that  
 1691 gradient-based optimization cannot learn the desired function from this initializa-  
 1692 tion.
- 1693 4. The simplest solution to the previous problem relies on the use of the ReLU activa-  
 1694 tion function at the hidden layer. Now consider a network with arbitrary activations  
 1695 on the hidden layer. Show that if the initial weights are any uniform constant, then  
 1696 it is not possible to learn the desired function.
- 1697 5. Consider a network in which: the base features are all binary,  $\mathbf{x} \in \{0, 1\}^M$ ; the  
 1698 hidden layer activation function is sigmoid,  $z_k = \sigma(\boldsymbol{\theta}_k \cdot \mathbf{x})$ ; and the initial weights  
 1699 are sampled independently from a standard normal distribution,  $\theta_{j,k} \sim N(0, 1)$ .

- 1700 • Show how the probability of a small initial gradient on any weight,  $\frac{\partial z_k}{\partial \theta_{j,k}} < \alpha$ ,  
 1701 depends on the size of the input  $M$ . **Hint:** use the lower bound,

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

1702 and relate this probability to the variance  $V[\boldsymbol{\theta}_k \cdot \mathbf{x}]$ .

- 1703 • Design an alternative initialization that removes this dependence.

- 1704 6. Suppose that the parameters  $\Theta = \{\Theta^{(x \rightarrow z)}, \Theta(z \rightarrow y), \mathbf{b}\}$  are a local optimum of a  
 1705 feedforward network in the following sense: there exists some  $\epsilon > 0$  such that,

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

1706 Define the function  $\pi$  as a permutation on the hidden units, as described in § 3.3.3,  
 1707 so that for any  $\Theta$ ,  $L(\Theta) = L(\Theta_\pi)$ . Prove that if a feedforward network has a local  
 optimum in the sense of Equation 3.61, then its loss is not a convex function of the  
 parameters  $\Theta$ , using the definition of convexity from § 2.3

- 1708     7. Consider a network with a single hidden layer, and a single output,

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

1709     Assume that  $g$  is the ReLU function. Prove that for any matrix of weights  $\Theta^{(x \rightarrow z)}$ , it  
1710     is permissible to rescale each row to have a norm of one, because an identical output  
1711     can be obtained by finding a corresponding rescaling of  $\theta^{(z \rightarrow y)}$ .

## <sup>1712</sup> Chapter 4

# <sup>1713</sup> Linguistic applications of <sup>1714</sup> classification

<sup>1715</sup> 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.

### <sup>1718</sup> 4.1 Sentiment and opinion analysis

<sup>1719</sup> A popular application of text classification is to automatically determine the **sentiment**  
<sup>1720</sup> or **opinion polarity** of documents such as product reviews and social media posts. For  
<sup>1721</sup> example, marketers are interested to know how people respond to advertisements, ser-  
<sup>1722</sup> vices, and products (Hu and Liu, 2004); social scientists are interested in how emotions  
<sup>1723</sup> are affected by phenomena such as the weather (Hannak et al., 2012), and how both opin-  
<sup>1724</sup> ions and emotions spread over social networks (Coviello et al., 2014; Miller et al., 2011).  
<sup>1725</sup> In the field of **digital humanities**, literary scholars track plot structures through the flow  
<sup>1726</sup> of sentiment across a novel (Jockers, 2015).<sup>1</sup>

<sup>1727</sup> Sentiment analysis can be framed as a direct application of document classification,  
<sup>1728</sup> assuming reliable labels can be obtained. In the simplest case, sentiment analysis is a  
<sup>1729</sup> two or three-class problem, with sentiments of POSITIVE, NEGATIVE, and possibly NEU-  
<sup>1730</sup> TRAL. Such annotations could be annotated by hand, or obtained automatically through  
<sup>1731</sup> a variety of means:

- <sup>1732</sup> • Tweets containing happy emoticons can be marked as positive, sad emoticons as  
<sup>1733</sup> negative (Read, 2005; Pak and Paroubek, 2010).

---

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

- 1734     • Reviews with four or more stars can be marked as positive, two or fewer stars as  
 1735       negative (Pang et al., 2002).
- 1736     • Statements from politicians who are voting for a given bill are marked as positive  
 1737       (towards that bill); statements from politicians voting against the bill are marked as  
 1738       negative (Thomas et al., 2006).

1739     The bag-of-words model is a good fit for sentiment analysis at the document level: if  
 1740     the document is long enough, we would expect the words associated with its true senti-  
 1741     ment to overwhelm the others. Indeed, **lexicon-based sentiment analysis** avoids machine  
 1742     learning altogether, and classifies documents by counting words against positive and neg-  
 1743     ative sentiment word lists (Taboada et al., 2011).

1744     Lexicon-based classification is less effective for short documents, such as single-sentence  
 1745     reviews or social media posts. In these documents, linguistic issues like **negation** and **ir-**  
 1746     **realis** (Polanyi and Zaenen, 2006) — events that are hypothetical or otherwise non-factual  
 1747     — can make bag-of-words classification ineffective. Consider the following examples:

- 1748     (4.1) That's not bad for the first day.
- 1749     (4.2) This is not the worst thing that can happen.
- 1750     (4.3) It would be nice if you acted like you understood.
- 1751     (4.4) There is no reason at all to believe that the polluters are suddenly going to be-  
 1752       come reasonable. (Wilson et al., 2005)
- 1753     (4.5) This film should be brilliant. The actors are first grade. Stallone plays a happy,  
 1754       wonderful man. His sweet wife is beautiful and adores him. He has a fascinat-  
 1755       ing gift for living life fully. It sounds like a great plot, **however**, the film is a  
 1756       failure. (Pang et al., 2002)

1757     A minimal solution is to move from a bag-of-words model to a bag-of-**bigrams** model,  
 1758     where each base feature is a pair of adjacent words, e.g.,

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

1759     Bigrams can handle relatively straightforward cases, such as when an adjective is immedi-  
 1760       ately negated; trigrams would be required to extend to larger contexts (e.g., *not the worst*).  
 1761     But this approach will not scale to more complex examples like (4.4) and (4.5). More  
 1762     sophisticated solutions try to account for the syntactic structure of the sentence (Wilson  
 1763     et al., 2005; Socher et al., 2013), or apply more complex classifiers such as **convolutional**  
 1764     **neural networks** (Kim, 2014), which are described in chapter 3.

1765 **4.1.1 Related problems**

1766 **Subjectivity** Closely related to sentiment analysis is **subjectivity detection**, which re-  
1767 quires identifying the parts of a text that express subjective opinions, as well as other non-  
1768 factual content such as speculation and hypotheticals (Riloff and Wiebe, 2003). This can be  
1769 done by treating each sentence as a separate document, and then applying a bag-of-words  
1770 classifier: indeed, Pang and Lee (2004) do exactly this, using a training set consisting of  
1771 (mostly) subjective sentences gathered from movie reviews, and (mostly) objective sen-  
1772 tences gathered from plot descriptions. They augment this bag-of-words model with a  
1773 graph-based algorithm that encourages nearby sentences to have the same subjectivity  
1774 label.

1775 **Stance classification** In debates, each participant takes a side: for example, advocating  
1776 for or against proposals like adopting a vegetarian lifestyle or mandating free college ed-  
1777 ucation. The problem of stance classification is to identify the author’s position from the  
1778 text of the argument. In some cases, there is training data available for each position,  
1779 so that standard document classification techniques can be employed. In other cases, it  
1780 suffices to classify each document as whether it is in support or opposition of the argu-  
1781 ment advanced by a previous document (Anand et al., 2011). In the most challenging  
1782 case, there is no labeled data for any of the stances, so the only possibility is group docu-  
1783 ments that advocate the same position (Somasundaran and Wiebe, 2009). This is a form  
1784 of **unsupervised learning**, discussed in chapter 5.

1785 **Targeted sentiment analysis** The expression of sentiment is often more nuanced than a  
1786 simple binary label. Consider the following examples:

1787 (4.6) The vodka was good, but the meat was rotten.

1788 (4.7) Go to Heaven for the climate, Hell for the company. —Mark Twain

1789 These statements display a mixed overall sentiment: positive towards some entities (e.g.,  
1790 *the vodka*), negative towards others (e.g., *the meat*). **Targeted sentiment analysis** seeks to  
1791 identify the writer’s sentiment towards specific entities (Jiang et al., 2011). This requires  
1792 identifying the entities in the text and linking them to specific sentiment words — much  
1793 more than we can do with the classification-based approaches discussed thus far. For  
1794 example, Kim and Hovy (2006) analyze sentence-internal structure to determine the topic  
1795 of each sentiment expression.

1796 **Aspect-based opinion mining** seeks to identify the sentiment of the author of a review  
1797 towards predefined aspects such as PRICE and SERVICE, or, in the case of (4.7), CLIMATE  
1798 and COMPANY (Hu and Liu, 2004). If the aspects are not defined in advance, it may again  
1799 be necessary to employ **unsupervised learning** methods to identify them (e.g., Branavan  
1800 et al., 2009).

1801 **Emotion classification** While sentiment analysis is framed in terms of positive and neg-  
 1802 ative categories, psychologists generally regard **emotion** as more multifaceted. For ex-  
 1803 ample, Ekman (1992) argues that there are six basic emotions — happiness, surprise, fear,  
 1804 sadness, anger, and contempt — and that they are universal across human cultures. Alm  
 1805 et al. (2005) build a linear classifier for recognizing the emotions expressed in children’s  
 1806 stories. The ultimate goal of this work was to improve text-to-speech synthesis, so that  
 1807 stories could be read with intonation that reflected the emotional content. They used bag-  
 1808 of-words features, as well as features capturing the story type (e.g., jokes, folktales), and  
 1809 structural features that reflect the position of each sentence in the story. The task is diffi-  
 1810 cult: even human annotators frequently disagreed with each other, and the best classifiers  
 1811 achieved accuracy between 60-70%.

#### 1812 4.1.2 Alternative approaches to sentiment analysis

1813 **Regression** A more challenging version of sentiment analysis is to determine not just  
 1814 the class of a document, but its rating on a numerical scale (Pang and Lee, 2005). If the  
 1815 scale is continuous, it is most natural to apply **regression**, identifying a set of weights  $\theta$   
 1816 that minimize the squared error of a predictor  $\hat{y} = \theta \cdot x + b$ , where  $b$  is an offset. This  
 1817 approach is called **linear regression**, and sometimes **least squares**, because the regression  
 1818 coefficients  $\theta$  are determined by minimizing the squared error,  $(y - \hat{y})^2$ . If the weights are  
 1819 regularized using a penalty  $\lambda \|\theta\|_2^2$ , then it is **ridge regression**. Unlike logistic regression,  
 1820 both linear regression and ridge regression can be solved in closed form as a system of  
 1821 linear equations.

1822 **Ordinal ranking** In many problems, the labels are ordered but discrete: for example,  
 1823 product reviews are often integers on a scale of 1 – 5, and grades are on a scale of A – F.  
 1824 Such problems can be solved by discretizing the score  $\theta \cdot x$  into “ranks”,

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

1825 where  $\mathbf{b} = [b_1 = -\infty, b_2, b_3, \dots, b_K]$  is a vector of boundaries. It is possible to learn the  
 1826 weights and boundaries simultaneously, using a perceptron-like algorithm (Crammer and  
 1827 Singer, 2001).

1828 **Lexicon-based classification** Sentiment analysis is one of the only NLP tasks where  
 1829 hand-crafted feature weights are still widely employed. In **lexicon-based classification** (Taboada  
 1830 et al., 2011), the user creates a list of words for each label, and then classifies each docu-  
 1831 ment based on how many of the words from each list are present. In our linear classifica-  
 1832 tion framework, this is equivalent to choosing the following weights:

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

1833 where  $\mathcal{L}_y$  is the lexicon for label  $y$ . Compared to the machine learning classifiers discussed  
 1834 in the previous chapters, lexicon-based classification may seem primitive. However, su-  
 1835 pervised machine learning relies on large annotated datasets, which are time-consuming  
 1836 and expensive to produce. If the goal is to distinguish two or more categories in a new  
 1837 domain, it may be simpler to start by writing down a list of words for each category.

1838 An early lexicon was the *General Inquirer* (Stone, 1966). Today, popular sentiment lex-  
 1839 cons include sentiwordnet (Esuli and Sebastiani, 2006) and an evolving set of lexicons  
 1840 from Liu (2015). For emotions and more fine-grained analysis, *Linguistic Inquiry and Word*  
 1841 *Count* (LIWC) provides a set of lexicons (Tausczik and Pennebaker, 2010). The MPQA lex-  
 1842 icon indicates the polarity (positive or negative) of 8221 terms, as well as whether they are  
 1843 strongly or weakly subjective (Wiebe et al., 2005). A comprehensive comparison of senti-  
 1844 ment lexicons is offered by Ribeiro et al. (2016). Given an initial **seed lexicon**, it is possible  
 1845 to automatically expand the lexicon by looking for words that frequently co-occur with  
 1846 words in the seed set (Hatzivassiloglou and McKeown, 1997; Qiu et al., 2011).

## 1847 4.2 Word sense disambiguation

1848 Consider the the following headlines:

- 1849 (4.8) Iraqi head seeks arms
- 1850 (4.9) Prostitutes appeal to Pope
- 1851 (4.10) Drunk gets nine years in violin case<sup>2</sup>

1852 These headlines are ambiguous because they contain words that have multiple mean-  
 1853 ings, or **senses**. Word sense disambiguation is the problem of identifying the intended  
 1854 sense of each word token in a document. Word sense disambiguation is part of a larger  
 1855 field of research called **lexical semantics**, which is concerned with meanings of the words.

1856 At a basic level, the problem of word sense disambiguation is to identify the correct  
 1857 sense for each word token in a document. Part-of-speech ambiguity (e.g., noun versus  
 1858 verb) is usually considered to be a different problem, to be solved at an earlier stage.  
 1859 From a linguistic perspective, senses are not properties of words, but of **lemmas**, which  
 1860 are canonical forms that stand in for a set of inflected words. For example, *arm*/N is a  
 1861 lemma that includes the inflected form *arms*/N — the /N indicates that it we are refer-  
 1862 ring to the noun, and not its **homonym** *arm*/V, which is another lemma that includes  
 1863 the inflected verbs (*arm*/V, *arms*/V, *armed*/V, *arming*/V). Therefore, word sense disam-  
 1864 biguation requires first identifying the correct part-of-speech and lemma for each token,

---

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

1865 and then choosing the correct sense from the inventory associated with the corresponding  
 1866 lemma.<sup>3</sup> (Part-of-speech tagging is discussed in § 8.1.)

1867 **4.2.1 How many word senses?**

1868 Words sometimes have many more than two senses, as exemplified by the word *serve*:

- 1869 • [FUNCTION]: *The tree stump served as a table*
- 1870 • [CONTRIBUTE TO]: *His evasive replies only served to heighten suspicion*
- 1871 • [PROVIDE]: *We serve only the rawest fish*
- 1872 • [ENLIST]: *She served in an elite combat unit*
- 1873 • [JAIL]: *He served six years for a crime he didn't commit*
- 1874 • [LEGAL]: *They were served with subpoenas*<sup>4</sup>

1875 These sense distinctions are annotated in **WordNet** (<http://wordnet.princeton.edu>), a lexical semantic database for English. WordNet consists of roughly 100,000 **synsets**,  
 1876 which are groups of lemmas (or phrases) that are synonymous. An example synset is  
 1877  $\{chump^1, fool^2, sucker^1, mark^9\}$ , where the superscripts index the sense of each lemma that  
 1878 is included in the synset: for example, there are at least eight other senses of *mark* that  
 1879 have different meanings, and are not part of this synset. A lemma is **polysemous** if it  
 1880 participates in multiple synsets.

1882 WordNet defines the scope of the word sense disambiguation problem, and, more  
 1883 generally, formalizes lexical semantic knowledge of English. (WordNets have been cre-  
 1884 ated for a few dozen other languages, at varying levels of detail.) Some have argued  
 1885 that WordNet's sense granularity is too fine (Ide and Wilks, 2006); more fundamentally,  
 1886 the premise that word senses can be differentiated in a task-neutral way has been criti-  
 1887 cized as linguistically naïve (Kilgarriff, 1997). One way of testing this question is to ask  
 1888 whether people tend to agree on the appropriate sense for example sentences: accord-  
 1889 ing to Mihalcea et al. (2004), people agree on roughly 70% of examples using WordNet  
 1890 senses; far better than chance, but less than agreement on other tasks, such as sentiment  
 1891 annotation (Wilson et al., 2005).

1892 **\*Other lexical semantic relations** Besides **synonymy**, WordNet also describes many  
 1893 other lexical semantic relationships, including:

- 1894 • **antonymy**: *x* means the opposite of *y*, e.g. FRIEND-ENEMY;

---

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

<sup>4</sup>Several of the examples are adapted from WordNet (Fellbaum, 2010).

- 1895 • **hyponymy:**  $x$  is a special case of  $y$ , e.g. RED-COLOR; the inverse relationship is  
1896      **hypernymy**;
- 1897 • **meronymy:**  $x$  is a part of  $y$ , e.g., WHEEL-BICYCLE; the inverse relationship is **holonymy**.

1898      Classification of these relations can be performed by searching for characteristic pat-  
1899      terns between pairs of words, e.g.,  $X$ , *such as*  $Y$ , which signals hyponymy (Hearst, 1992),  
1900      or  $X$  *but*  $Y$ , which signals antonymy (Hatzivassiloglou and McKeown, 1997). Another ap-  
1901      proach is to analyze each term's **distributional statistics** (the frequency of its neighboring  
1902      words). Such approaches are described in detail in chapter 14.

### 1903 4.2.2 Word sense disambiguation as classification

1904 How can we tell living *plants* from manufacturing *plants*? The context is often critical:

- 1905 (4.11) Town officials are hoping to attract new manufacturing plants through weakened  
1906      environmental regulations.
- 1907 (4.12) The endangered plants play an important role in the local ecosystem.

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

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

1908 As in document classification, many of these features are irrelevant, but a few are very  
1909 strong predictors. In this example, the context word *endangered* is a strong signal that  
1910 the intended sense is biology rather than manufacturing. We would therefore expect a  
1911 learning algorithm to assign high weight to (*endangered*, BIOLOGY), and low weight to  
1912 (*endangered*, MANUFACTURING).<sup>5</sup>

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

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

1913 These are called **collocation features**, and they give more information about the specific  
1914 role played by each context word. This idea can be taken further by incorporating addi-  
1915 tional syntactic information about the grammatical role played by each context feature,  
1916 such as the **dependency path** (see chapter 11).

---

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

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

(4.13) As of Sunday<sup>1</sup><sub>N</sub> night<sup>1</sup><sub>N</sub> there was<sup>4</sup><sub>V</sub> no word<sup>2</sup><sub>N</sub> ...,

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

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

### 4.3 Design decisions for text classification

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

#### 4.3.1 What is a word?

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

##### 4.3.1.1 Tokenization

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

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

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

1950 to define a subset of characters as whitespace, and then split the text on these tokens.  
 1951 However, whitespace-based tokenization is not ideal: we may want to split conjunctions  
 1952 like *isn't* and hyphenated phrases like *prize-winning* and *half-asleep*, and we likely want  
 1953 to separate words from commas and periods that immediately follow them. At the same  
 1954 time, it would be better not to split abbreviations like *U.S.* and *Ph.D.* In languages with  
 1955 Roman scripts, tokenization is typically performed using regular expressions, with mod-  
 1956 ules designed to handle each of these cases. For example, the `nltk` package includes a  
 1957 number of tokenizers (Loper and Bird, 2002); the outputs of four of the better-known tok-  
 1958 enizers are shown in Figure 4.1. Social media researchers have found that emoticons and  
 1959 other forms of orthographic variation pose new challenges for tokenization, leading to the  
 1960 development of special purpose tokenizers to handle these phenomena (O'Connor et al.,  
 1961 2010).

1962 Tokenization is a language-specific problem, and each language poses unique chal-  
 1963 lenges. For example, Chinese does not include spaces between words, nor any other  
 1964 consistent orthographic markers of word boundaries. A “greedy” approach is to scan the  
 1965 input for character substrings that are in a predefined lexicon. However, Xue et al. (2003)  
 1966 notes that this can be ambiguous, since many character sequences could be segmented in  
 1967 multiple ways. Instead, he trains a classifier to determine whether each Chinese character,  
 1968 or *hanzi*, is a word boundary. More advanced sequence labeling methods for word seg-  
 1969 mentation are discussed in § 8.4. Similar problems can occur in languages with alphabetic  
 1970 scripts, such as German, which does not include whitespace in compound nouns, yield-  
 1971 ing examples such as *Freundschaftsbezeugungen* (demonstration of friendship) and *Dilett-*  
*1972 tantenaufdringlichkeiten* (the importunities of dilettantes). As Twain (1997) argues, “*These*  
*1973 things are not words, they are alphabetic processions.*” Social media raises similar problems  
 1974 for English and other languages, with hashtags such as *#TrueLoveInFourWords* requiring  
 1975 decomposition for analysis (Brun and Roux, 2014).

### 1976 4.3.1.2 Normalization

1977 After splitting the text into tokens, the next question is which tokens are really distinct.  
 1978 Is it necessary to distinguish *great*, *Great*, and *GREAT*? Sentence-initial capitalization may  
 1979 be irrelevant to the classification task. Going further, the complete elimination of case  
 1980 distinctions will result in a smaller vocabulary, and thus smaller feature vectors. However,

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

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

1981 case distinctions might be relevant in some situations: for example, *apple* is a delicious  
 1982 pie filling, while *Apple* is a company that specializes in proprietary dongles and power  
 1983 adapters.

1984 For Roman script, case conversion can be performed using unicode string libraries.  
 1985 Many scripts do not have case distinctions (e.g., the Devanagari script used for South  
 1986 Asian languages, the Thai alphabet, and Japanese kana), and case conversion for all scripts  
 1987 may not be available in every programming environment. (Unicode support is an im-  
 1988 portant distinction between Python’s versions 2 and 3, and is a good reason for mi-  
 1989 grating to Python 3 if you have not already done so. Compare the output of the code  
 1990 "\à l\'hôtel".upper() in the two language versions.)<sup>6</sup>

1991 Case conversion is a type of **normalization**, which refers to string transformations that  
 1992 remove distinctions that are irrelevant to downstream applications (Sproat et al., 2001).  
 1993 Other normalizations include the standardization of numbers (e.g., 1,000 to 1000) and  
 1994 dates (e.g., August 11, 2015 to 2015/11/08). Depending on the application, it may even be  
 1995 worthwhile to convert all numbers and dates to special tokens, !NUM and !DATE. In social  
 1996 media, there are additional orthographic phenomena that may be normalized, such as ex-  
 1997 pressive lengthening, e.g., *coooooool* (Aw et al., 2006; Yang and Eisenstein, 2013). Similarly,  
 1998 historical texts feature spelling variations that may need to be normalized to a contempo-  
 1999 rary standard form (Baron and Rayson, 2008).

2000 A more extreme form of normalization is to eliminate **inflectional affixes**, such as the  
 2001 -ed and -s suffixes in English. On this view, *bike*, *bikes*, *biking*, and *biked* all refer to the  
 2002 same underlying concept, so they should be grouped into a single feature. A **stemmer** is  
 2003 a program for eliminating affixes, usually by applying a series of regular expression sub-  
 2004 stitutions. Character-based stemming algorithms are necessarily approximate, as shown  
 2005 in Figure 4.2: the Lancaster stemmer incorrectly identifies -ers as an inflectional suffix of  
 2006 *sisters* (by analogy to *fix/fixer*s), and both stemmers incorrectly identify -s as a suffix of *this*  
 2007 and *Williams*. Fortunately, even inaccurate stemming can improve bag-of-words classifi-  
 2008 cation models, by merging related strings and thereby reducing the vocabulary size.

2009 Accurately handling irregular orthography requires word-specific rules. **Lemmatizers**

---

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

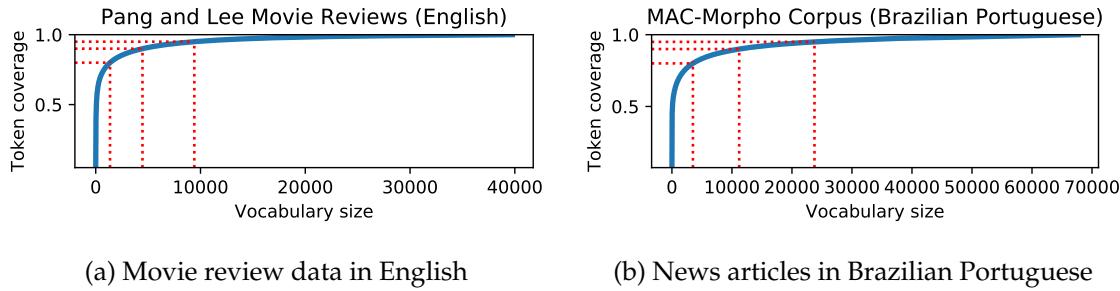


Figure 4.3: Tradeoff between token coverage (y-axis) and vocabulary size, on the `nltk` movie review dataset, after sorting the vocabulary by decreasing frequency. The red dashed lines indicate 80%, 90%, and 95% coverage.

2010 are systems that identify the underlying lemma of a given wordform. They must avoid the  
 2011 over-generalization errors of the stemmers in Figure 4.2, and also handle more complex  
 2012 transformations, such as *geese*→*goose*. The output of the WordNet lemmatizer is shown in  
 2013 the final line of Figure 4.2. Both stemming and lemmatization are language-specific: an  
 2014 English stemmer or lemmatizer is of little use on a text written in another language. The  
 2015 discipline of **morphology** relates to the study of word-internal structure, and is described  
 2016 in more detail in § 9.1.2.

2017 The value of normalization depends on the data and the task. Normalization re-  
 2018 duces the size of the feature space, which can help in generalization. However, there  
 2019 is always the risk of merging away linguistically meaningful distinctions. In supervised  
 2020 machine learning, regularization and smoothing can play a similar role to normalization  
 2021 — preventing the learner from overfitting to rare features — while avoiding the language-  
 2022 specific engineering required for accurate normalization. In unsupervised scenarios, such  
 2023 as content-based information retrieval (Manning et al., 2008) and topic modeling (Blei  
 2024 et al., 2003), normalization is more critical.

### 2025 4.3.2 How many words?

2026 Limiting the size of the feature vector reduces the memory footprint of the resulting mod-  
 2027 els, and increases the speed of prediction. Normalization can help to play this role, but  
 2028 a more direct approach is simply to limit the vocabulary to the  $N$  most frequent words  
 2029 in the dataset. For example, in the movie-reviews dataset provided with `nltk` (orig-  
 2030 inally from Pang et al., 2002), there are 39,768 word types, and 1.58M tokens. As shown  
 2031 in Figure 4.3a, the most frequent 4000 word types cover 90% of all tokens, offering an  
 2032 order-of-magnitude reduction in the model size. Such ratios are language-specific: in for  
 2033 example, in the Brazilian Portuguese Mac-Morpho corpus (Aluísio et al., 2003), attain-  
 2034 ing 90% coverage requires more than 10000 word types (Figure 4.3b). This reflects the

2035 morphological complexity of Portuguese, which includes many more inflectional suffixes  
 2036 than English.

2037 Eliminating rare words is not always advantageous for classification performance: for  
 2038 example, names, which are typically rare, play a large role in distinguishing topics of news  
 2039 articles. Another way to reduce the size of the feature space is to eliminate **stopwords** such  
 2040 as *the*, *to*, and *and*, which may seem to play little role in expressing the topic, sentiment,  
 2041 or stance. This is typically done by creating a **stoplist** (e.g., `nltk.corpus.stopwords`),  
 2042 and then ignoring all terms that match the list. However, corpus linguists and social psy-  
 2043 chologists have shown that seemingly inconsequential words can offer surprising insights  
 2044 about the author or nature of the text (Biber, 1991; Chung and Pennebaker, 2007). Further-  
 2045 more, high-frequency words are unlikely to cause overfitting in discriminative classifiers.  
 2046 As with normalization, stopword filtering is more important for unsupervised problems,  
 2047 such as term-based document retrieval.

2048 Another alternative for controlling model size is **feature hashing** (Weinberger et al.,  
 2049 2009). Each feature is assigned an index using a hash function. If a hash function that  
 2050 permits collisions is chosen (typically by taking the hash output modulo some integer),  
 2051 then the model can be made arbitrarily small, as multiple features share a single weight.  
 2052 Because most features are rare, accuracy is surprisingly robust to such collisions (Ganchev  
 2053 and Dredze, 2008).

### 2054 4.3.3 Count or binary?

2055 Finally, we may consider whether we want our feature vector to include the *count* of each  
 2056 word, or its *presence*. This gets at a subtle limitation of linear classification: it worse to  
 2057 have two *failures* than one, but is it really twice as bad? Motivated by this intuition, Pang  
 2058 et al. (2002) use binary indicators of presence or absence in the feature vector:  $f_j(x, y) \in$   
 2059  $\{0, 1\}$ . They find that classifiers trained on these binary vectors tend to outperform feature  
 2060 vectors based on word counts. One explanation is that words tend to appear in clumps:  
 2061 if a word has appeared once in a document, it is likely to appear again (Church, 2000).  
 2062 These subsequent appearances can be attributed to this tendency towards repetition, and  
 2063 thus provide little additional information about the class label of the document.

## 2064 4.4 Evaluating classifiers

2065 In any supervised machine learning application, it is critical to reserve a held-out test set.  
 2066 This data should be used for only one purpose: to evaluate the overall accuracy of a single  
 2067 classifier. Using this data more than once would cause the estimated accuracy to be overly  
 2068 optimistic, because the classifier would be customized to this data, and would not perform  
 2069 as well as on unseen data in the future. It is usually necessary to set hyperparameters or

2070 perform feature selection, so you may need to construct a **tuning** or **development set** for  
 2071 this purpose, as discussed in § 2.1.5.

2072 There are a number of ways to evaluate classifier performance. The simplest is **accuracy**:  
 2073 the number of correct predictions, divided by the total number of instances,

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

2074 Exams are usually graded by accuracy. Why are other metrics necessary? The main  
 2075 reason is **class imbalance**. Suppose you are building a classifier to detect whether an  
 2076 electronic health record (EHR) describes symptoms of a rare disease, which appears in  
 2077 only 1% of all documents in the dataset. A classifier that reports  $\hat{y} = \text{NEGATIVE}$  for  
 2078 all documents would achieve 99% accuracy, but would be practically useless. We need  
 2079 metrics that are capable of detecting the classifier's ability to discriminate between classes,  
 2080 even when the distribution is skewed.

2081 One solution is to build a **balanced test set**, in which each possible label is equally rep-  
 2082 resented. But in the EHR example, this would mean throwing away 98% of the original  
 2083 dataset! Furthermore, the detection threshold itself might be a design consideration: in  
 2084 health-related applications, we might prefer a very sensitive classifier, which returned a  
 2085 positive prediction if there is even a small chance that  $y^{(i)} = \text{POSITIVE}$ . In other applica-  
 2086 tions, a positive result might trigger a costly action, so we would prefer a classifier that  
 2087 only makes positive predictions when absolutely certain. We need additional metrics to  
 2088 capture these characteristics.

#### 2089 4.4.1 Precision, recall, and F-MEASURE

2090 For any label (e.g., positive for presence of symptoms of a disease), there are two possible  
 2091 errors:

- 2092 • **False positive**: the system incorrectly predicts the label.
- 2093 • **False negative**: the system incorrectly fails to predict the label.

2094 Similarly, for any label, there are two ways to be correct:

- 2095 • **True positive**: the system correctly predicts the label.
- 2096 • **True negative**: the system correctly predicts that the label does not apply to this  
 2097 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**:

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

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

2098 Recall and precision are both conditional likelihoods of a correct prediction, which is why  
 2099 their numerators are the same. Recall is conditioned on  $k$  being the correct label,  $y^{(i)} = k$ ,  
 2100 so the denominator sums over true positive and false negatives. Precision is conditioned  
 2101 on  $k$  being the prediction, so the denominator sums over true positives and false positives.  
 2102 Note that true negatives are not considered in either statistic. The classifier that labels  
 2103 every document as “negative” would achieve zero recall; precision would be  $\frac{0}{0}$ .

2104 Recall and precision are complementary. A high-recall classifier is preferred when  
 2105 false negatives are cheaper than false positives: for example, in a preliminary screening  
 2106 for symptoms of a disease, the cost of a false positive might be an additional test, while a  
 2107 false negative would result in the disease going untreated. Conversely, a high-precision  
 2108 classifier is preferred when false positives are more expensive: for example, in spam de-  
 2109tection, a false negative is a relatively minor inconvenience, while a false positive might  
 2110 mean that an important message goes unread.

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

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

2111 where  $r$  is recall and  $p$  is precision.<sup>7</sup>

**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,

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

2112 In multi-class problems with unbalanced class distributions, the macro  **$F$ -MEASURE** is a  
 2113 balanced measure of how well the classifier recognizes each class. In **micro  $F$ -MEASURE**,  
 2114 we compute true positives, false positives, and false negatives for each class, and then add  
 2115 them up to compute a single recall, precision, and  **$F$ -MEASURE**. This metric is balanced  
 2116 across instances rather than classes, so it weights each class in proportion to its frequency  
 2117 — unlike macro  **$F$ -MEASURE**, which weights each class equally.

---

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

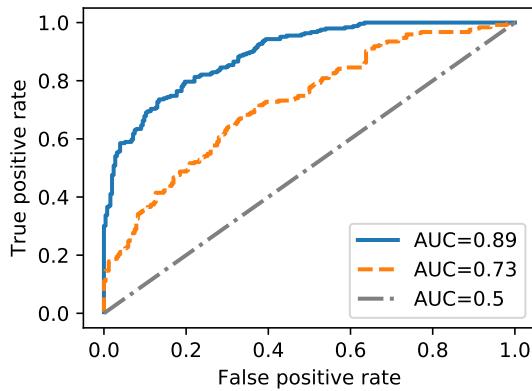


Figure 4.4: ROC curves for three classifiers of varying discriminative power, measured by AUC (area under the curve)

#### 2118 4.4.2 Threshold-free metrics

2119 In binary classification problems, it is possible to trade off between recall and precision by  
 2120 adding a constant “threshold” to the output of the scoring function. This makes it possible  
 2121 to trace out a curve, where each point indicates the performance at a single threshold. In  
 2122 the **receiver operating characteristic (ROC)** curve,<sup>8</sup> the *x*-axis indicates the **false positive**  
 2123 **rate**,  $\frac{FP}{FP+TN}$ , and the *y*-axis indicates the recall, or **true positive rate**. A perfect classifier  
 2124 attains perfect recall without any false positives, tracing a “curve” from the origin (0,0) to  
 2125 the upper left corner (0,1), and then to (1,1). In expectation, a non-discriminative classifier  
 2126 traces a diagonal line from the origin (0,0) to the upper right corner (1,1). Real classifiers  
 2127 tend to fall between these two extremes. Examples are shown in Figure 4.4.

2128 The ROC curve can be summarized in a single number by taking its integral, the **area**  
 2129 **under the curve (AUC)**. The AUC can be interpreted as the probability that a randomly-  
 2130 selected positive example will be assigned a higher score by the classifier than a randomly-  
 2131 selected negative example. A perfect classifier has AUC = 1 (all positive examples score  
 2132 higher than all negative examples); a non-discriminative classifier has AUC = 0.5 (given  
 2133 a randomly selected positive and negative example, either could score higher with equal  
 2134 probability); a perfectly wrong classifier would have AUC = 0 (all negative examples score  
 2135 higher than all positive examples). One advantage of AUC in comparison to *F*-MEASURE  
 2136 is that the baseline rate of 0.5 does not depend on the label distribution.

---

<sup>8</sup>The 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.

2137 **4.4.3 Classifier comparison and statistical significance**

2138 Natural language processing research and engineering often involves comparing different  
 2139 classification techniques. In some cases, the comparison is between algorithms, such as  
 2140 logistic regression versus averaged perceptron, or  $L_2$  regularization versus  $L_1$ . In other  
 2141 cases, the comparison is between feature sets, such as the bag-of-words versus positional  
 2142 bag-of-words (see § 4.2.2). **Ablation testing** involves systematically removing (ablating)  
 2143 various aspects of the classifier, such as feature groups, and testing the **null hypothesis**  
 2144 that the ablated classifier is as good as the full model.

2145 A full treatment of hypothesis testing is beyond the scope of this text, but this section  
 2146 contains a brief summary of the techniques necessary to compare classifiers. The main  
 2147 aim of hypothesis testing is to determine whether the difference between two statistics  
 2148 — for example, the accuracies of two classifiers — is likely to arise by chance. We will  
 2149 be concerned with chance fluctuations that arise due to the finite size of the test set.<sup>9</sup> An  
 2150 improvement of 10% on a test set with ten instances may reflect a random fluctuation that  
 2151 makes the test set more favorable to classifier  $c_1$  than  $c_2$ ; on another test set with a different  
 2152 ten instances, we might find that  $c_2$  does better than  $c_1$ . But if we observe the same 10%  
 2153 improvement on a test set with 1000 instances, this is highly unlikely to be explained  
 2154 by chance. Such a finding is said to be **statistically significant** at a level  $p$ , which is the  
 2155 probability of observing an effect of equal or greater magnitude when the null hypothesis  
 2156 is true. The notation  $p < .05$  indicates that the likelihood of an equal or greater effect is  
 2157 less than 5%, assuming the null hypothesis is true.<sup>10</sup>

2158 **4.4.3.1 The binomial test**

2159 The statistical significance of a difference in accuracy can be evaluated using classical tests,  
 2160 such as the **binomial test**.<sup>11</sup> Suppose that classifiers  $c_1$  and  $c_2$  disagree on  $N$  instances in a  
 2161 test set with binary labels, and that  $c_1$  is correct on  $k$  of those instances. Under the null hy-  
 2162 pothesis that the classifiers are equally accurate, we would expect  $k/N$  to be roughly equal  
 2163 to 1/2, and as  $N$  increases,  $k/N$  should be increasingly close to this expected value. These  
 2164 properties are captured by the **binomial distribution**, which is a probability over counts

---

<sup>9</sup>Other 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.

<sup>10</sup>Statistical 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.

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

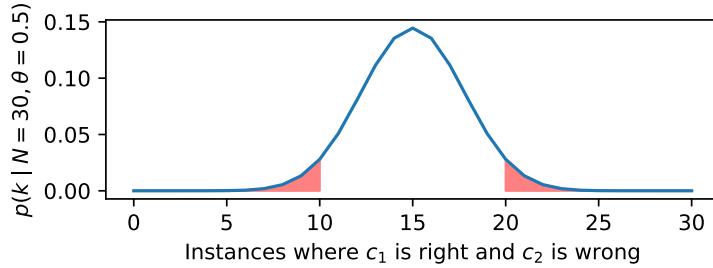


Figure 4.5: Probability mass function for the binomial distribution. The pink highlighted areas represent the cumulative probability for a significance test on an observation of  $k = 10$  and  $N = 30$ .

of binary random variables. We write  $k \sim \text{Binom}(\theta, N)$  to indicate that  $k$  is drawn from a binomial distribution, with parameter  $N$  indicating the number of random “draws”, and  $\theta$  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  $\theta^k$  representing the probability of the  $k$  successes,  $(1 - \theta)^{N-k}$  representing the probability of the  $N - k$  unsuccessful draws. The expression  $\binom{N}{k} = \frac{N!}{k!(N-k)!}$  is a binomial coefficient, representing the number of possible orderings of events; this ensures that the distribution sums to one over all  $k \in \{0, 1, 2, \dots, N\}$ .

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

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

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

---

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

---

```

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

---

2179 rate, we would take the sum of one-tailed  $p$ -values, where the second term is computed  
 2180 from the right tail of Figure 4.5. The binomial distribution is symmetric, so this can be  
 2181 computed by simply doubling the one-tailed  $p$ -value.

2182 Two-tailed tests are more stringent, but they are necessary in cases in which there is  
 2183 no prior intuition about whether  $c_1$  or  $c_2$  is better. For example, in comparing logistic  
 2184 regression versus averaged perceptron, a two-tailed test is appropriate. In an ablation  
 2185 test,  $c_2$  may contain a superset of the features available to  $c_1$ . If the additional features are  
 2186 thought to be likely to improve performance, then a one-tailed test would be appropriate,  
 2187 if chosen in advance. However, such a test can only prove that  $c_2$  is more accurate than  
 2188  $c_1$ , and not the reverse.

2189 **4.4.3.2 \*Randomized testing**

2190 The binomial test is appropriate for accuracy, but not for more complex metrics such as  
 2191  $F$ -MEASURE. To compute statistical significance for arbitrary metrics, we can apply ran-  
 2192 domization. Specifically, draw a set of  $M$  **bootstrap samples** (Efron and Tibshirani, 1993),  
 2193 by resampling instances from the original test set with replacement. Each bootstrap sam-  
 2194 ple is itself a test set of size  $N$ . Some instances from the original test set will not appear  
 2195 in any given bootstrap sample, while others will appear multiple times; but overall, the  
 2196 sample will be drawn from the same distribution as the original test set. We can then com-  
 2197 pute any desired evaluation on each bootstrap sample, which gives a distribution over the  
 2198 value of the metric. Algorithm 7 shows how to perform this computation.

2199 To compare the  $F$ -MEASURE of two classifiers  $c_1$  and  $c_2$ , we set the function  $\delta(\cdot)$  to  
 2200 compute the difference in  $F$ -MEASURE on the bootstrap sample. If the difference is less  
 2201 than or equal to zero in at least 5% of the samples, then we cannot reject the one-tailed  
 2202 null hypothesis that  $c_2$  is at least as good as  $c_1$  (Berg-Kirkpatrick et al., 2012). We may  
 2203 also be interested in the 95% **confidence interval** around a metric of interest, such as  
 2204 the  $F$ -MEASURE of a single classifier. This can be computed by sorting the output of

2205 Algorithm 7, and then setting the top and bottom of the 95% confidence interval to the  
 2206 values at the 2.5% and 97.5% percentiles of the sorted outputs. Alternatively, you can fit  
 2207 a normal distribution to the set of differences across bootstrap samples, and compute a  
 2208 Gaussian confidence interval from the mean and variance.

2209 As the number of bootstrap samples goes to infinity,  $M \rightarrow \infty$ , the bootstrap estimate  
 2210 is increasingly accurate. A typical choice for  $M$  is  $10^4$  or  $10^5$ ; larger numbers of samples  
 2211 are necessary for smaller  $p$ -values. One way to validate your choice of  $M$  is to run the test  
 2212 multiple times, and ensure that the  $p$ -values are similar; if not, increase  $M$  by an order of  
 2213 magnitude. This is a heuristic measure of the **variance** of the test, which can decrease  
 2214 with the square root  $\sqrt{M}$  (Robert and Casella, 2013).

#### 2215 4.4.4 \*Multiple comparisons

2216 Sometimes it is necessary to perform multiple hypothesis tests, such as when compar-  
 2217 ing the performance of several classifiers on multiple datasets. Suppose you have five  
 2218 datasets, and you compare four versions of your classifier against a baseline system, for a  
 2219 total of 20 comparisons. Even if none of your classifiers is better than the baseline, there  
 2220 will be some chance variation in the results, and in expectation you will get one statis-  
 2221 tically significant improvement at  $p = 0.05 = \frac{1}{20}$ . It is therefore necessary to adjust the  
 2222  $p$ -values when reporting the results of multiple comparisons.

2223 One approach is to require a threshold of  $\frac{\alpha}{m}$  to report a  $p$  value of  $p < \alpha$  when per-  
 2224 forming  $m$  tests. This is known as the **Bonferroni correction**, and it limits the overall  
 2225 probability of incorrectly rejecting the null hypothesis at  $\alpha$ . Another approach is to bound  
 2226 the **false discovery rate** (FDR), which is the fraction of null hypothesis rejections that are  
 2227 incorrect. Benjamini and Hochberg (1995) propose a  $p$ -value correction that bounds the  
 2228 fraction of false discoveries at  $\alpha$ : sort the  $p$ -values of each individual test in ascending  
 2229 order, and set the significance threshold equal to largest  $k$  such that  $p_k \leq \frac{k}{m}\alpha$ . If  $k > 1$ , the  
 2230 FDR adjustment is more permissive than the Bonferroni correction.

## 2231 4.5 Building datasets

2232 Sometimes, if you want to build a classifier, you must first build a dataset of your own.  
 2233 This includes selecting a set of documents or instances to annotate, and then performing  
 2234 the annotations. The scope of the dataset may be determined by the application: if you  
 2235 want to build a system to classify electronic health records, then you must work with a  
 2236 corpus of records of the type that your classifier will encounter when deployed. In other  
 2237 cases, the goal is to build a system that will work across a broad range of documents. In  
 2238 this case, it is best to have a *balanced* corpus, with contributions from many styles and  
 2239 genres. For example, the Brown corpus draws from texts ranging from government doc-  
 2240 uments to romance novels (Francis, 1964), and the Google Web Treebank includes an-

2241 notations for five “domains” of web documents: question answers, emails, newsgroups,  
2242 reviews, and blogs (Petrov and McDonald, 2012).

2243 **4.5.1 Metadata as labels**

2244 Annotation is difficult and time-consuming, and most people would rather avoid it. It  
2245 is sometimes possible to exploit existing metadata to obtain labels for training a classi-  
2246 fier. For example, reviews are often accompanied by a numerical rating, which can be  
2247 converted into a classification label (see § 4.1). Similarly, the nationalities of social media  
2248 users can be estimated from their profiles (Dredze et al., 2013) or even the time zones of  
2249 their posts (Gouws et al., 2011). More ambitiously, we may try to classify the political af-  
2250 filiations of social media profiles based on their social network connections to politicians  
2251 and major political parties (Rao et al., 2010).

2252 The convenience of quickly constructing large labeled datasets without manual an-  
2253 notation is appealing. However this approach relies on the assumption that unlabeled  
2254 instances — for which metadata is unavailable — will be similar to labeled instances.  
2255 Consider the example of labeling the political affiliation of social media users based on  
2256 their network ties to politicians. If a classifier attains high accuracy on such a test set,  
2257 is it safe to assume that it accurately predicts the political affiliation of all social media  
2258 users? Probably not. Social media users who establish social network ties to politicians  
2259 may be more likely to mention politics in the text of their messages, as compared to the  
2260 average user, for whom no political metadata is available. If so, the accuracy on a test set  
2261 constructed from social network metadata would give an overly optimistic picture of the  
2262 method’s true performance on unlabeled data.

2263 **4.5.2 Labeling data**

2264 In many cases, there is no way to get ground truth labels other than manual annotation.  
2265 An annotation protocol should satisfy several criteria: the annotations should be *expressive*  
2266 enough to capture the phenomenon of interest; they should be *replicable*, meaning that  
2267 another annotator or team of annotators would produce very similar annotations if given  
2268 the same data; and they should be *scalable*, so that they can be produced relatively quickly.  
2269 Hovy and Lavid (2010) propose a structured procedure for obtaining annotations that  
2270 meet these criteria, which is summarized below.

- 2271 1. **Determine what the annotations are to include.** This is usually based on some  
2272 theory of the underlying phenomenon: for example, if the goal is to produce an-  
2273 notations about the emotional state of a document’s author, one should start with a  
2274 theoretical account of the types or dimensions of emotion (e.g., Mohammad and Tur-  
2275 ney, 2013). At this stage, the tradeoff between expressiveness and scalability should

2276 be considered: a full instantiation of the underlying theory might be too costly to  
2277 annotate at scale, so reasonable approximations should be considered.

- 2278 2. Optionally, one may **design or select a software tool to support the annotation**  
2279 **effort**. Existing general-purpose annotation tools include BRAT (Stenetorp et al.,  
2280 2012) and MMAX2 (Müller and Strube, 2006).
- 2281 3. **Formalize the instructions for the annotation task.** To the extent that the instruc-  
2282 tions are not explicit, the resulting annotations will depend on the intuitions of the  
2283 annotators. These intuitions may not be shared by other annotators, or by the users  
2284 of the annotated data. Therefore explicit instructions are critical to ensuring the an-  
2285 notations are replicable and usable by other researchers.
- 2286 4. **Perform a pilot annotation** of a small subset of data, with multiple annotators for  
2287 each instance. This will give a preliminary assessment of both the replicability and  
2288 scalability of the current annotation instructions. Metrics for computing the rate of  
2289 agreement are described below. Manual analysis of specific disagreements should  
2290 help to clarify the instructions, and may lead to modifications of the annotation task  
2291 itself. For example, if two labels are commonly conflated by annotators, it may be  
2292 best to merge them.
- 2293 5. **Annotate the data.** After finalizing the annotation protocol and instructions, the  
2294 main annotation effort can begin. Some, if not all, of the instances should receive  
2295 multiple annotations, so that inter-annotator agreement can be computed. In some  
2296 annotation projects, instances receive many annotations, which are then aggregated  
2297 into a “consensus” label (e.g., Danescu-Niculescu-Mizil et al., 2013). However, if the  
2298 annotations are time-consuming or require significant expertise, it may be preferable  
2299 to maximize scalability by obtaining multiple annotations for only a small subset of  
2300 examples.
- 2301 6. **Compute and report inter-annotator agreement, and release the data.** In some  
2302 cases, the raw text data cannot be released, due to concerns related to copyright or  
2303 privacy. In these cases, one solution is to publicly release **stand-off annotations**,  
2304 which contain links to document identifiers. The documents themselves can be re-  
2305 leased under the terms of a licensing agreement, which can impose conditions on  
2306 how the data is used. It is important to think through the potential consequences of  
2307 releasing data: people may make personal data publicly available without realizing  
2308 that it could be redistributed in a dataset and publicized far beyond their expecta-  
2309 tions (boyd and Crawford, 2012).

2310 **4.5.2.1 Measuring inter-annotator agreement**

2311 To measure the replicability of annotations, a standard practice is to compute the extent to  
 2312 which annotators agree with each other. If the annotators frequently disagree, this casts  
 2313 doubt on either their reliability or on the annotation system itself. For classification, one  
 2314 can compute the frequency with which the annotators agree; for rating scales, one can  
 2315 compute the average distance between ratings. These raw agreement statistics must then  
 2316 be compared with the rate of **chance agreement** — the level of agreement that would be  
 2317 obtained between two annotators who ignored the data.

2318 **Cohen's Kappa** is widely used for quantifying the agreement on discrete labeling  
 2319 tasks (Cohen, 1960; Carletta, 1996),<sup>12</sup>

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

2320 The numerator is the difference between the observed agreement and the chance agree-  
 2321 ment, and the denominator is the difference between perfect agreement and chance agree-  
 2322 ment. Thus,  $\kappa = 1$  when the annotators agree in every case, and  $\kappa = 0$  when the annota-  
 2323 tors agree only as often as would happen by chance. Various heuristic scales have been  
 2324 proposed for determining when  $\kappa$  indicates "moderate", "good", or "substantial" agree-  
 2325 ment; for reference, Lee and Narayanan (2005) report  $\kappa \approx 0.45 - 0.47$  for annotations  
 2326 of emotions in spoken dialogues, which they describe as "moderate agreement"; Stolcke  
 2327 et al. (2000) report  $\kappa = 0.8$  for annotations of **dialogue acts**, which are labels for the pur-  
 2328 pose of each turn in a conversation.

2329 When there are two annotators, the expected chance agreement is computed as,

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

2330 where  $k$  is a sum over labels, and  $\hat{\Pr}(Y = k)$  is the empirical probability of label  $k$  across  
 2331 all annotations. The formula is derived from the expected number of agreements if the  
 2332 annotations were randomly shuffled. Thus, in a binary labeling task, if one label is applied  
 2333 to 90% of instances, chance agreement is  $.9^2 + .1^2 = .82$ .

2334 **4.5.2.2 Crowdsourcing**

2335 Crowdsourcing is often used to rapidly obtain annotations for classification problems.  
 2336 For example, **Amazon Mechanical Turk** makes it possible to define "human intelligence  
 2337 tasks (hits)", such as labeling data. The researcher sets a price for each set of annotations  
 2338 and a list of minimal qualifications for annotators, such as their native language and their

---

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

2339 satisfaction rate on previous tasks. The use of relatively untrained “crowdworkers” con-  
 2340 trasts with earlier annotation efforts, which relied on professional linguists (Marcus et al.,  
 2341 1993). However, crowdsourcing has been found to produce reliable annotations for many  
 2342 language-related tasks (Snow et al., 2008). Crowdsourcing is part of the broader field of  
 2343 **human computation** (Law and Ahn, 2011).

## 2344 Additional resources

2345 Many of the preprocessing issues discussed in this chapter also arise in information re-  
 2346 trieval. See (Manning et al., 2008, chapter 2) for discussion of tokenization and related  
 2347 algorithms.

## 2348 Exercises

2349 1. As noted in § 4.3.3, words tend to appear in clumps, with subsequent occurrences  
 2350 of a word being more probable. More concretely, if word  $j$  has probability  $\phi_{y,j}$   
 2351 of appearing in a document with label  $y$ , then the probability of two appearances  
 2352 ( $x_j^{(i)} = 2$ ) is greater than  $\phi_{y,j}^2$ .

2353 Suppose you are applying Naïve Bayes to a binary classification. Focus on a word  $j$   
 2354 which is more probable under label  $y = 1$ , so that,

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

2355 Now suppose that  $x_j^{(i)} > 1$ . All else equal, will the classifier overestimate or under-  
 2356 estimate the posterior  $\Pr(y = 1 \mid x)$ ?

2357 2. Prove that F-measure is never greater than the arithmetic mean of recall and pre-  
 2358 cision,  $\frac{r+p}{2}$ . Your solution should also show that F-measure is equal to  $\frac{r+p}{2}$  iff  $r = p$ .

2359 3. Given a binary classification problem in which the probability of the “positive” label  
 2360 is equal to  $\alpha$ , what is the expected F-MEASURE of a random classifier which ignores  
 2361 the data, and selects  $\hat{y} = +1$  with probability  $\frac{1}{2}$ ? (Assume that  $p(\hat{y}) \perp p(y)$ .) What is  
 2362 the expected F-MEASURE of a classifier that selects  $\hat{y} = +1$  with probability  $\alpha$  (also  
 2363 independent of  $y^{(i)}$ )? Depending on  $\alpha$ , which random classifier will score better?

2364 4. Suppose that binary classifiers  $c_1$  and  $c_2$  disagree on  $N = 30$  cases, and that  $c_1$  is  
 2365 correct in  $k = 10$  of those cases.

- 2366 • Write a program that uses primitive functions such as `exp` and `factorial` to com-  
 2367 pute the **two-tailed**  $p$ -value — you may use an implementation of the “choose”  
 2368 function if one is available. Verify your code against the output of a library for

2369 computing the binomial test or the binomial CDF, such as `scipy.stats.binom`  
 2370 in Python.

- 2371 • Then use a randomized test to try to obtain the same  $p$ -value. In each sample,  
 2372 draw from a binomial distribution with  $N = 30$  and  $\theta = \frac{1}{2}$ . Count the fraction  
 2373 of samples in which  $k \leq 10$ . This is the one-tailed  $p$ -value; double this to  
 2374 compute the two-tailed  $p$ -value.
- 2375 • Try this with varying numbers of bootstrap samples:  $M \in \{100, 1000, 5000, 10000\}$ .  
 2376 For  $M = 100$  and  $M = 1000$ , run the test 10 times, and plot the resulting  $p$ -  
 2377 values.
- 2378 • Finally, perform the same tests for  $N = 70$  and  $k = 25$ .

2379 5. SemCor 3.0 is a labeled dataset for word sense disambiguation. You can download  
 2380 it,<sup>13</sup> or access it in `nltk.corpora.semcor`.

2381 Choose a word that appears at least ten times in SemCor (*find*), and annotate its  
 2382 WordNet senses across ten randomly-selected examples, without looking at the ground  
 2383 truth. Use online WordNet to understand the definition of each of the senses.<sup>14</sup> Have  
 2384 a partner do the same annotations, and compute the raw rate of agreement, expected  
 2385 chance rate of agreement, and Cohen's kappa.

2386 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  
 2387 randomly-selected 400 reviews as a test set.

2388 Download a sentiment lexicon, such as the one currently available from Bing Liu,  
 2389 <https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html>. Tokenize  
 2390 the data, and classify each document as positive iff it has more positive sentiment  
 2391 words than negative sentiment words. Compute the accuracy and *F*-MEASURE on  
 2392 detecting positive reviews on the test set, using this lexicon-based classifier.

2393 Then train a discriminative classifier (averaged perceptron or logistic regression) on  
 2394 the training set, and compute its accuracy and *F*-MEASURE on the test set.

2395 Determine whether the differences are statistically significant, using two-tailed hy-  
 2396 pothesis tests: Binomial for the difference in accuracy, and bootstrap for the differ-  
 2397 ence in macro-*F*-MEASURE.

2398 2399 The remaining problems will require you to build a classifier and test its properties. Pick  
 2400 a multi-class text classification dataset, such as RCV1<sup>15</sup>). Divide your data into training

---

<sup>13</sup>e.g., [https://github.com/google-research-datasets/word\\_sense\\_disambiguation\\_corpora](https://github.com/google-research-datasets/word_sense_disambiguation_corpora) or <http://globalwordnet.org/wordnet-annotated-corpora/>

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

<sup>15</sup>[http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/lyrl2004\\_rcv1v2\\_README.htm](http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm)

2401 (60%), development (20%), and test sets (20%), if no such division already exists. [todo:  
2402 this dataset is already tokenized, find something else]

2403 7. Compare various vocabulary sizes of  $10^2, 10^3, 10^4, 10^5$ , using the most frequent words  
2404 in each case (you may use any reasonable tokenizer). Train logistic regression clas-  
2405 sifiers for each vocabulary size, and apply them to the development set. Plot the  
2406 accuracy and Macro-*F*-MEASURE with the increasing vocabulary size. For each vo-  
2407 cabulary size, tune the regularizer to maximize accuracy on a subset of data that is  
2408 held out from the training set.

2409 8. Compare the following tokenization algorithms:

- 2410 • Whitespace, using a regular expression  
2411 • Penn Treebank  
2412 • Split input into five-character units, regardless of whitespace or punctuation

2413 Compute the token/type ratio for each tokenizer on the training data, and explain  
2414 what you find. Train your classifier on each tokenized dataset, tuning the regularizer  
2415 on a subset of data that is held out from the training data. Tokenize the development  
2416 set, and report accuracy and Macro-*F*-MEASURE.

2417 9. Apply the Porter and Lancaster stemmers to the training set, using any reasonable  
2418 tokenizer, and compute the token/type ratios. Train your classifier on the stemmed  
2419 data, and compute the accuracy and Macro-*F*-MEASURE on stemmed development  
2420 data, again using a held-out portion of the training data to tune the regularizer.

2421 10. Identify the best combination of vocabulary filtering, tokenization, and stemming  
2422 from the previous three problems. Apply this preprocessing to the test set, and  
2423 compute the test set accuracy and Macro-*F*-MEASURE. Compare against a baseline  
2424 system that applies no vocabulary filtering, whitespace tokenization, and no stem-  
2425 ming.

2426 Use the binomial test to determine whether your best-performing system is signifi-  
2427 cantly more accurate than the baseline.

2428 Use the bootstrap test with  $M = 10^4$  to determine whether your best-performing  
2429 system achieves significantly higher macro-*F*-MEASURE.



# 2430 Chapter 5

## 2431 Learning without supervision

2432 So far we've assumed the following setup:

- 2433 a **training set** where you get observations  $x$  and labels  $y$ ;
- 2434 a **test set** where you only get observations  $x$ .

2435 Without labeled data, is it possible to learn anything? This scenario is known as **unsu-**  
2436 **pervised learning**, and we will see that indeed it is possible to learn about the underlying  
2437 structure of unlabeled observations. This chapter will also explore some related scenarios:  
2438 **semi-supervised learning**, in which only some instances are labeled, and **domain adap-**  
2439 **tation**, in which the training data differs from the data on which the trained system will  
2440 be deployed.

### 2441 5.1 Unsupervised learning

2442 To motivate unsupervised learning, consider the problem of word sense disambiguation  
2443 (§ 4.2). Our goal is to classify each instance of a word, such as *bank* into a sense,

- 2444 bank#1: a financial institution
- 2445 bank#2: the land bordering a river

2446 It is difficult to obtain sufficient training data for word sense disambiguation, because  
2447 even a large corpus will contain only a few instances of all but the most common words.  
2448 Is it possible to learn anything about these different senses without labeled data?

2449 Word sense disambiguation is usually performed using feature vectors constructed  
2450 from the local context of the word to be disambiguated. For example, for the word

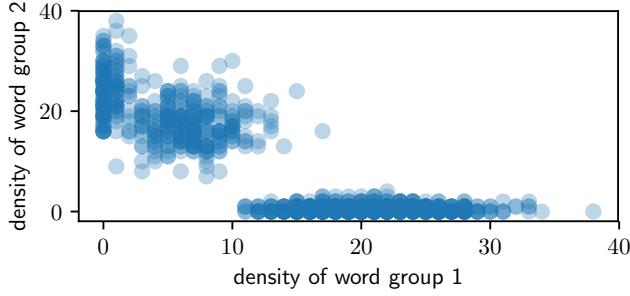


Figure 5.1: Counts of words from two different context groups

2451 *bank*, the immediate context might typically include words from one of the following two  
 2452 groups:

- 2453 1. *financial, deposits, credit, lending, capital, markets, regulated, reserve, liquid, assets*  
 2454 2. *land, water, geography, stream, river, flow, deposits, discharge, channel, ecology*

2455 Now consider a scatterplot, in which each point is a document containing the word *bank*.  
 2456 The location of the document on the  $x$ -axis is the count of words in group 1, and the  
 2457 location on the  $y$ -axis is the count for group 2. In such a plot, shown in Figure 5.1, two  
 2458 “blobs” might emerge, and these blobs correspond to the different senses of *bank*.

2459 Here’s a related scenario, from a different problem. Suppose you download thousands  
 2460 of news articles, and make a scatterplot, where each point corresponds to a document:  
 2461 the  $x$ -axis is the frequency of the group of words (*hurricane, winds, storm*); the  $y$ -axis is the  
 2462 frequency of the group (*election, voters, vote*). This time, three blobs might emerge: one  
 2463 for documents that are largely about a hurricane, another for documents largely about a  
 2464 election, and a third for documents about neither topic.

2465 These clumps represent the underlying structure of the data. But the two-dimensional  
 2466 scatter plots are based on groupings of context words, and in real scenarios these word  
 2467 lists are unknown. Unsupervised learning applies the same basic idea, but in a high-  
 2468 dimensional space with one dimension for every context word. This space can’t be di-  
 2469 rectly visualized, but the idea is the same: try to identify the underlying structure of the  
 2470 observed data, such that there are a few clusters of points, each of which is internally  
 2471 coherent. **Clustering** algorithms are capable of finding such structure automatically.

### 2472 5.1.1 **K-means** clustering

2473 Clustering algorithms assign each data point to a discrete cluster,  $z_i \in 1, 2, \dots, K$ . One of  
 2474 the best known clustering algorithms is ***K-means***, an iterative algorithm that maintains

**Algorithm 8**  $K$ -means clustering algorithm

---

```

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

```

---

2475 a cluster assignment for each instance, and a central (“mean”) location for each cluster.  
 2476  $K$ -means iterates between updates to the assignments and the centers:

- 2477 1. each instance is placed in the cluster with the closest center;  
 2478 2. each center is recomputed as the average over points in the cluster.

2479 This is formalized in Algorithm 8. The term  $\|\mathbf{x}^{(i)} - \boldsymbol{\nu}\|^2$  refers to the squared Euclidean  
 2480 norm,  $\sum_{j=1}^V (x_j^{(i)} - \nu_j)^2$ .

2481 **Soft  $K$ -means** is a particularly relevant variant. Instead of directly assigning each  
 2482 point to a specific cluster, soft  $K$ -means assigns each point a **distribution** over clusters  
 2483  $\mathbf{q}^{(i)}$ , so that  $\sum_{k=1}^K q^{(i)}(k) = 1$ , and  $\forall_k, q^{(i)}(k) \geq 0$ . The soft weight  $q^{(i)}(k)$  is computed from  
 2484 the distance of  $\mathbf{x}^{(i)}$  to the cluster center  $\boldsymbol{\nu}_k$ . In turn, the center of each cluster is computed  
 2485 from a **weighted average** of the points in the cluster,

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

2486 We will now explore a probabilistic version of soft  $K$ -means clustering, based on **expectation**  
 2487 **maximization** (EM). Because EM clustering can be derived as an approximation to  
 2488 maximum-likelihood estimation, it can be extended in a number of useful ways.

2489 **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  $\mathbf{x}$ :

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

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

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

2490 To estimate the parameters  $\boldsymbol{\phi}$  and  $\boldsymbol{\mu}$ , we can maximize the marginal likelihood in Equa-  
 2491 tion 5.5. Why is this the right thing to maximize? Without labels, discriminative learning  
 2492 is impossible — there's nothing to discriminate. So maximum likelihood is all we have.

2493 When the labels are observed, we can estimate the parameters of the Naïve Bayes  
 2494 probability model separately for each label. But marginalizing over the labels couples  
 2495 these parameters, making direct optimization of  $\log p(\mathbf{x})$  intractable. We will approximate  
 2496 the log-likelihood by introducing an *auxiliary variable*  $\mathbf{q}^{(i)}$ , which is a distribution over the  
 2497 label set  $\mathcal{Z} = \{1, 2, \dots, K\}$ . The optimization procedure will alternate between updates to  
 2498  $\mathbf{q}$  and updates to the parameters  $(\boldsymbol{\phi}, \boldsymbol{\mu})$ . Thus,  $\mathbf{q}^{(i)}$  plays here as in soft  $K$ -means.

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

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

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

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

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

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

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

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

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

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

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

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

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  $q^{(i)}$ :

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

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

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

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

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

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

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

### 5.1.2.2 The M-step

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

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

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

2517 The term  $\log p(\mathbf{x}^{(i)} | z; \boldsymbol{\phi})$  is the conditional log-likelihood for the multinomial, which  
2518 expands to,

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

2519 where  $C$  is a constant with respect to  $\boldsymbol{\phi}$  — see Equation 2.12 in § 2.1 for more discussion  
2520 of this probability function.

Setting the derivative of  $J_\phi$  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  $\phi_z$  is constrained to be a probability distribution, the exact solution is computed as,

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

2521 where the counter  $j \in \{1, 2, \dots, V\}$  indexes over base features, such as words.

2522 This update sets  $\phi_z$  equal to the relative frequency estimate of the *expected counts* under  
2523 the distribution  $q$ . As in supervised Naïve Bayes, we can smooth these counts by adding  
2524 a constant  $\alpha$ . The update for  $\mu$  is similar:  $\mu_z \propto \sum_{i=1}^N q^{(i)}(z) = E_q [\text{count}(z)]$ , which is the  
2525 expected frequency of cluster  $z$ . These probabilities can also be smoothed. In sum, the  
2526 M-step is just like Naïve Bayes, but with expected counts rather than observed counts.

2527 The multinomial likelihood  $p(\mathbf{x} | z)$  can be replaced with other probability distribu-  
2528 tions: for example, for continuous observations, a Gaussian distribution can be used. In  
2529 some cases, there is no closed-form update to the parameters of the likelihood. One ap-  
2530 proach is to run gradient-based optimization at each M-step; another is to simply take a  
2531 single step along the gradient step and then return to the E-step (Berg-Kirkpatrick et al.,  
2532 2010).

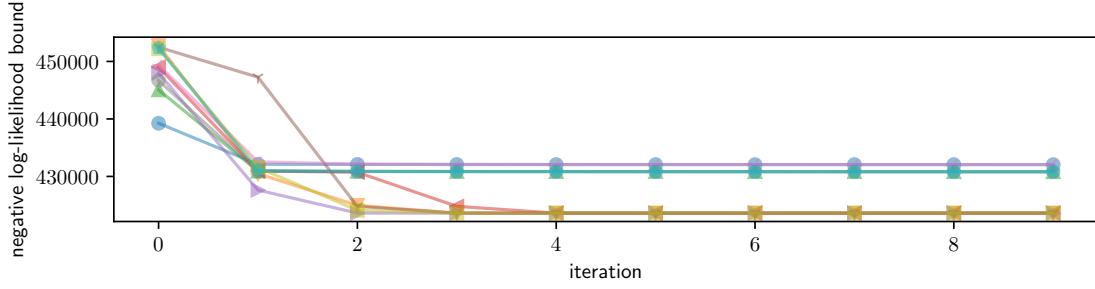


Figure 5.2: Sensitivity of expectation maximization to initialization. Each line shows the progress of optimization from a different random initialization.

### 2533 5.1.3 EM as an optimization algorithm

2534 Algorithms that alternate between updating subsets of the parameters are called **coordi-**  
 2535 **nate ascent** algorithms. The objective  $J$  (the lower bound on the marginal likelihood of  
 2536 the data) is separately convex in  $q$  and  $(\mu, \phi)$ , but it is not jointly convex in all terms; this  
 2537 condition is known as **biconvexity**. Each step of the expectation-maximization algorithm  
 2538 is guaranteed not to decrease the lower bound  $J$ , which means that EM will converge  
 2539 towards a solution at which no nearby points yield further improvements. This solution  
 2540 is a **local optimum** — it is as good or better than any of its immediate neighbors, but is  
 2541 *not* guaranteed to be optimal among all possible configurations of  $(q, \mu, \phi)$ .

2542 The fact that there is no guarantee of global optimality means that initialization is  
 2543 important: where you start can determine where you finish. To illustrate this point,  
 2544 Figure 5.2 shows the objective function for EM with ten different random initializations:  
 2545 while the objective function improves monotonically in each run, it converges to several  
 2546 different values.<sup>1</sup> For the convex objectives that we encountered in chapter 2, it was not  
 2547 necessary to worry about initialization, because gradient-based optimization guaranteed  
 2548 to reach the global minimum. But in expectation-maximization — and in the deep neural  
 2549 networks from chapter 3 — initialization matters.

2550 In **hard EM**, each  $q^{(i)}$  distribution assigns probability of 1 to a single label  $\hat{z}^{(i)}$ , and zero  
 2551 probability to all others (Neal and Hinton, 1998). This is similar in spirit to  $K$ -means clus-  
 2552 tering, and can outperform standard EM in some cases (Spitkovsky et al., 2010). Another  
 2553 variant of expectation maximization incorporates stochastic gradient descent (SGD): after  
 2554 performing a local E-step at each instance  $x^{(i)}$ , we immediately make a gradient update  
 2555 to the parameters  $(\mu, \phi)$ . This algorithm has been called **incremental expectation maxi-**  
 2556 **mization** (Neal and Hinton, 1998) and **online expectation maximization** (Sato and Ishii,  
 2557 2000; Cappé and Moulines, 2009), and is especially useful when there is no closed-form

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

2558 optimum for the likelihood  $p(\mathbf{x} \mid z)$ , and in online settings where new data is constantly  
 2559 streamed in (see Liang and Klein, 2009, for a comparison for online EM variants).

2560 **5.1.4 How many clusters?**

2561 So far, we have assumed that the number of clusters  $K$  is given. In some cases, this as-  
 2562 sumption is valid. For example, a lexical semantic resource like WordNet might define the  
 2563 number of senses for a word. In other cases, the number of clusters could be a parameter  
 2564 for the user to tune: some readers want a coarse-grained clustering of news stories into  
 2565 three or four clusters, while others want a fine-grained clustering into twenty or more.  
 2566 But many times there is little extrinsic guidance for how to choose  $K$ .

2567 One solution is to choose the number of clusters to maximize a metric of clustering  
 2568 quality. The other parameters  $\mu$  and  $\phi$  are chosen to maximize the log-likelihood bound  
 2569  $J$ , so this might seem a potential candidate for tuning  $K$ . However,  $J$  will never decrease  
 2570 with  $K$ : if it is possible to obtain a bound of  $J_K$  with  $K$  clusters, then it is always possible  
 2571 to do at least as well with  $K + 1$  clusters, by simply ignoring the additional cluster and  
 2572 setting its probability to zero in  $q$  and  $\mu$ . It is therefore necessary to introduce a penalty  
 2573 for model complexity, so that fewer clusters are preferred. For example, the Akaike Infor-  
 2574 mation Crition (AIC; Akaike, 1974) is the linear combination of the number of parameters  
 2575 and the log-likelihood,

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

2576 where  $M$  is the number of parameters. In an expectation-maximization clustering algo-  
 2577 rithm,  $M = K \times V + K$ . Since the number of parameters increases with the number of  
 2578 clusters  $K$ , the AIC may prefer more parsimonious models, even if they do not fit the data  
 2579 quite as well.

2580 Another choice is to maximize the **predictive likelihood** on heldout data. This data  
 2581 is not used to estimate the model parameters  $\phi$  and  $\mu$ , and so it is not the case that the  
 2582 likelihood on this data is guaranteed to increase with  $K$ . Figure 5.3 shows the negative  
 2583 log-likelihood on training and heldout data, as well as the AIC.

2584 \***Bayesian nonparametrics** An alternative approach is to treat the number of clusters as  
 2585 another latent variable. This requires statistical inference over a set of models with a vari-  
 2586 able number of clusters. This is not possible within the framework of expectation max-  
 2587 imization, but there are several alternative inference procedures which can be applied,  
 2588 including **Markov Chain Monte Carlo (MCMC)**, which is briefly discussed in § 5.5 (for  
 2589 more details, see Chapter 25 of Murphy, 2012). Bayesian nonparametrics have been ap-  
 2590 plied to the problem of unsupervised word sense induction, learning not only the word  
 2591 senses but also the number of senses per word (Reisinger and Mooney, 2010).

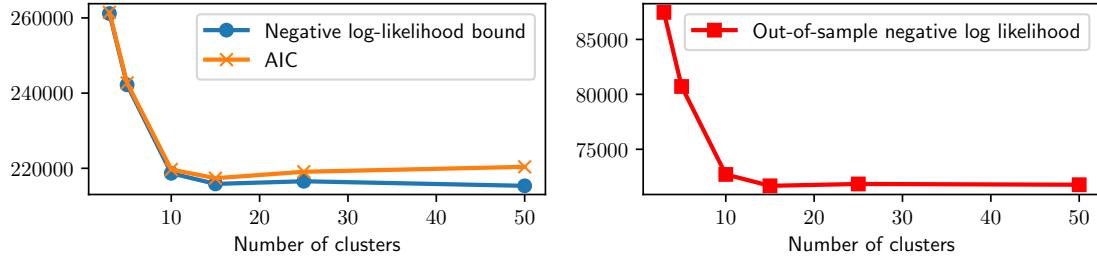


Figure 5.3: The negative log-likelihood and AIC for several runs of expectation maximization, on synthetic data. Although the data was generated from a model with  $K = 10$ , the optimal number of clusters is  $\hat{K} = 15$ , according to AIC and the heldout log-likelihood. The training set log-likelihood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

## 2592 5.2 Applications of expectation-maximization

2593 EM is not really an “algorithm” like, say, quicksort. Rather, it is a framework for learning  
2594 with missing data. The recipe for using EM on a problem of interest is:

- 2595 • Introduce latent variables  $z$ , such that it is easy to write the probability  $P(x, z)$ . It  
2596 should also be easy to estimate the associated parameters, given knowledge of  $z$ .
- 2597 • Derive the E-step updates for  $q(z)$ , which is typically factored as  $q(z) = \prod_{i=1}^N q_{z(i)}(z^{(i)})$ ,  
2598 where  $i$  is an index over instances.
- 2599 • The M-step updates typically correspond to the soft version of a probabilistic super-  
2600 vised learning algorithm, like Naïve Bayes.

2601 This section discusses a few of the many applications of this general framework.

### 2602 5.2.1 Word sense induction

2603 The chapter began by considering the problem of word sense disambiguation when the  
2604 senses are not known in advance. Expectation-maximization can be applied to this prob-  
2605 lem by treating each cluster as a word sense. Each instance represents the use of an  
2606 ambiguous word, and  $x^{(i)}$  is a vector of counts for the other words that appear nearby:  
2607 Schütze (1998) uses all words within a 50-word window. The probability  $p(x^{(i)} | z)$  can be  
2608 set to the multinomial distribution, as in Naïve Bayes. The EM algorithm can be applied  
2609 directly to this data, yielding clusters that (hopefully) correspond to the word senses.

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

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

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

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

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

## 5.2.2 Semi-supervised learning

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

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

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

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

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

---

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

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

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

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

---

2629 The left sum is identical to the objective in Naïve Bayes; the right sum is the marginal log-  
 2630 likelihood for expectation-maximization clustering, from Equation 5.5. We can construct a  
 2631 lower bound on this log-likelihood by introducing distributions  $q^{(j)}$  for all  $j \in \{N_\ell + 1, \dots, N_\ell + N_u\}$ .  
 2632 The E-step updates these distributions; the M-step updates the parameters  $\phi$  and  $\mu$ , us-  
 2633 ing the expected counts from the unlabeled data and the observed counts from the labeled  
 2634 data.

2635 A critical issue in semi-supervised learning is how to balance the impact of the labeled  
 2636 and unlabeled data on the classifier weights, especially when the unlabeled data is much  
 2637 larger than the labeled dataset. The risk is that the unlabeled data will dominate, caus-  
 2638 ing the parameters to drift towards a “natural clustering” of the instances — which may  
 2639 not correspond to a good classifier for the labeled data. One solution is to heuristically  
 2640 reweight the two components of Equation 5.26, tuning the weight of the two components  
 2641 on a heldout development set (Nigam et al., 2000).

2642 **5.2.3 Multi-component modeling**

2643 As a final application, let’s return to fully supervised classification. A classic dataset for  
 2644 text classification is 20 newsgroups, which contains posts to a set of online forums, called  
 2645 newsgroups. One of the newsgroups is `comp.sys.mac.hardware`, which discusses Ap-  
 2646 ple computing hardware. Suppose that within this newsgroup there are two kinds of  
 2647 posts: reviews of new hardware, and question-answer posts about hardware problems.  
 2648 The language in these *components* of the `mac.hardware` class might have little in com-  
 2649 mon; if so, it would be better to model these components separately, rather than treating  
 2650 their union as a single class. However, the component responsible for each instance is not  
 2651 directly observed.

2652 Recall that Naïve Bayes is based on a generative process, which provides a stochastic  
 2653 explanation for the observed data. In Naïve Bayes, each label is drawn from a categorical  
 2654 distribution with parameter  $\mu$ , and each vector of word counts is drawn from a multi-  
 2655 nomial distribution with parameter  $\phi_y$ . For multi-component modeling, we envision a  
 2656 slightly different generative process, incorporating both the observed label  $y^{(i)}$  and the  
 2657 latent component  $z^{(i)}$ . This generative process is shown in Algorithm 9. A new parameter  
 2658  $\beta_{y^{(i)}}$  defines the distribution of components, conditioned on the label  $y^{(i)}$ . The component,  
 2659 and not the class label, then parametrizes the distribution over words.

- 
- (5.1) ☺ Villeneuve a bel et bien **réussi** son pari de changer de perspectives tout en assurant une cohérence à la franchise.<sup>2</sup>
- (5.2) ☺ Il est également trop **long** et bancal dans sa narration, tiède dans ses intentions, et tirailé entre deux personnages et directions qui ne parviennent pas à coexister en harmonie.<sup>3</sup>
- (5.3) Denis Villeneuve a **réussi** une suite **parfaitemment** maîtrisée<sup>4</sup>
- (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.<sup>5</sup>
- (5.5) Une suite d'une écrasante puissance, mêlant **parfaitemment** le contemplatif au narratif.<sup>6</sup>
- (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.<sup>7</sup>

---

Table 5.1: Labeled and unlabeled reviews of the films *Blade Runner 2049* and *Transformers: The Last Knight*.

The labeled data includes  $(\mathbf{x}^{(i)}, y^{(i)})$ , but not  $z^{(i)}$ , so this is another case of missing data. Again, we sum over the missing data, applying Jensen's inequality to as to obtain a lower bound on the log-likelihood,

$$\log p(\mathbf{x}^{(i)}, y^{(i)}) = \log \sum_{z=1}^{K_z} p(\mathbf{x}^{(i)}, y^{(i)}, z; \boldsymbol{\mu}, \boldsymbol{\phi}, \boldsymbol{\beta}) \quad [5.28]$$

$$\geq \log p(y^{(i)}; \boldsymbol{\mu}) + E_{q_{Z|Y}^{(i)}} [\log p(\mathbf{x}^{(i)} | z; \boldsymbol{\phi}) + \log p(z | y^{(i)}; \boldsymbol{\beta}) - \log q^{(i)}(z)]. \quad [5.29]$$

We are now ready to apply expectation maximization. As usual, the E-step updates the distribution over the missing data,  $q_{Z|Y}^{(i)}$ . The M-step updates the parameters,

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

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

## 2660 5.3 Semi-supervised learning

2661 In semi-supervised learning, the learner makes use of both labeled and unlabeled data.  
 2662 To see how this could help, suppose you want to do sentiment analysis in French. In Ta-

ble 5.1, there are two labeled examples, one positive and one negative. From this data, a learner could conclude that *réussi* is positive and *long* is negative. This isn't much! However, we can propagate this information to the unlabeled data, and potentially learn more.

- If we are confident that *réussi* is positive, then we might guess that (5.3) is also positive.
- That suggests that *parfaitement* is also positive.
- We can then propagate this information to (5.5), and learn from this words in this example.
- Similarly, we can propagate from the labeled data to (5.4), which we guess to be negative because it shares the word *long*. This suggests that *bavard* is also negative, which we propagate to (5.6).

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

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

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

### 5.3.1 Multi-view learning

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

	$\mathbf{x}^{(1)}$	$\mathbf{x}^{(2)}$	$y$
1.	Peachtree Street	located on	LOC
2.	Dr. Walker	said	PER
3.	Zanzibar	located in	? → LOC
4.	Zanzibar	flew to	? → LOC
5.	Dr. Robert	recommended	? → PER
6.	Oprah	recommended	? → PER

Table 5.2: Example of multiview learning for named entity classification

2699     **Co-training** is an iterative multi-view learning algorithm, in which there are separate  
 2700 classifiers for each view (Blum and Mitchell, 1998). At each iteration of the algorithm, each  
 2701 classifier predicts labels for a subset of the unlabeled instances, using only the features  
 2702 available in its view. These predictions are then used as ground truth to train the classifiers  
 2703 associated with the other views. In the example shown in Table 5.2, the classifier on  $\mathbf{x}^{(1)}$   
 2704 might correctly label instance #5 as a person, because of the feature *Dr*; this instance would  
 2705 then serve as training data for the classifier on  $\mathbf{x}^{(2)}$ , which would then be able to correctly  
 2706 label instance #6, thanks to the feature *recommended*. If the views are truly independent,  
 2707 this procedure is robust to drift. Furthermore, it imposes no restrictions on the classifiers  
 2708 that can be used for each view.

2709     Word-sense disambiguation is particularly suited to multi-view learning, thanks to the  
 2710 heuristic of “one sense per discourse”: if a polysemous word is used more than once in  
 2711 a given text or conversation, all usages refer to the same sense (Gale et al., 1992). This  
 2712 motivates a multi-view learning approach, in which one view corresponds to the local  
 2713 context (the surrounding words), and another view corresponds to the global context at  
 2714 the document level (Yarowsky, 1995). The local context view is first trained on a small  
 2715 seed dataset. We then identify its most confident predictions on unlabeled instances. The  
 2716 global context view is then used to extend these confident predictions to other instances  
 2717 within the same documents. These new instances are added to the training data to the  
 2718 local context classifier, which is retrained and then applied to the remaining unlabeled  
 2719 data.

### 2720 5.3.2 Graph-based algorithms

2721     Another family of approaches to semi-supervised learning begins by constructing a graph,  
 2722 in which pairs of instances are linked with symmetric weights  $\omega_{i,j}$ , e.g.,

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

2723     The goal is to use this weighted graph to propagate labels from a small set of labeled  
 2724 instances to larger set of unlabeled instances.

2725 In **label propagation**, this is done through a series of matrix operations (Zhu et al.,  
 2726 Let  $\mathbf{Q}$  be a matrix of size  $N \times K$ , in which each row  $\mathbf{q}^{(i)}$  describes the labeling  
 2727 of instance  $i$ . When ground truth labels are available, then  $\mathbf{q}^{(i)}$  is an indicator vector,  
 2728 with  $q_{y^{(i)}}^{(i)} = 1$  and  $q_{y' \neq y^{(i)}}^{(i)} = 0$ . Let us refer to the submatrix of rows containing labeled  
 2729 instances as  $\mathbf{Q}_L$ , and the remaining rows as  $\mathbf{Q}_U$ . The rows of  $\mathbf{Q}_U$  are initialized to assign  
 2730 equal probabilities to all labels,  $q_{i,k} = \frac{1}{K}$ .

2731 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  $\kappa$  largest values in each row, and setting all other values to zero. We can then “propagate” the label distributions to the unlabeled instances,

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

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

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

2732 The expression  $\tilde{\mathbf{Q}}_U \mathbf{1}$  indicates multiplication of  $\tilde{\mathbf{Q}}_U$  by a column vector of ones, which is  
 2733 equivalent to computing the sum of each row of  $\tilde{\mathbf{Q}}_U$ . The matrix  $\text{Diag}(\mathbf{s})$  is a diagonal  
 2734 matrix with the elements of  $\mathbf{s}$  on the diagonals. The product  $\text{Diag}(\mathbf{s})^{-1} \tilde{\mathbf{Q}}_U$  has the effect  
 2735 of normalizing the rows of  $\tilde{\mathbf{Q}}_U$ , so that each row of  $\mathbf{Q}_U$  is a probability distribution over  
 2736 labels.

## 2737 5.4 Domain adaptation

2738 In many practical scenarios, the labeled data differs in some key respect from the data  
 2739 to which the trained model is to be applied. A classic example is in consumer reviews:  
 2740 we may have labeled reviews of movies (the **source domain**), but we want to predict the  
 2741 reviews of appliances (the **target domain**). A similar issues arise with genre differences:  
 2742 most linguistically-annotated data is news text, but application domains range from social  
 2743 media to electronic health records. In general, there may be several source and target  
 2744 domains, each with their own properties; however, for simplicity, this discussion will  
 2745 focus mainly on the case of a single source and target domain.

2746 The simplest approach is “direct transfer”: train a classifier on the source domain,  
 2747 and apply it directly to the target domain. The accuracy of this approach depends on the  
 2748 extent to which features are shared across domains. In review text, words like *outstanding*

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

#### 5.4.1 Supervised domain adaptation

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

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

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

where  $\Psi_s$  and  $\Psi_t$  are the scoring functions from the source and target domain classifiers respectively, and  $\lambda_s$  is the interpolation weight.

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

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

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

where  $\ell^{(i)}$  is the prediction loss on instance  $i$ , and  $\lambda$  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....*<sup>8</sup> The resulting bag-of-words feature

---

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

vector would be,

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

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

#### 5.4.2 Unsupervised domain adaptation

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

##### 5.4.2.1 Linear projection

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

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

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

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

---

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

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

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

#### 5.4.2.2 Non-linear projection

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

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

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

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

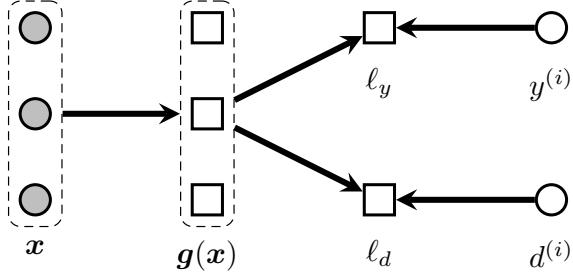


Figure 5.4: A schematic view of adversarial domain adaptation. The loss  $\ell_y$  is computed only for instances from the source domain, where labels  $y^{(i)}$  are available.

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

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

where  $N_\ell$  is the number of labeled instances and  $N_u$  is the number of unlabeled instances, with the labeled instances appearing first in the dataset. This setup is shown in Figure 5.4. The loss can be optimized by stochastic gradient descent, jointly training the parameters of the non-linear transformation  $\theta_g$ , and the parameters of the prediction models  $\theta_d$  and  $\theta_y$ .

## 5.5 \*Other approaches to learning with latent variables

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

### 5.5.1 Sampling

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

2854 data  $\mathbf{z}^{(1:N_z)}$  is the set of cluster memberships,  $\mathbf{y}^{(1:N)}$ , so we draw samples from the pos-  
 2855 terior distribution over clusterings of the data. If a single clustering is required, we can  
 2856 select the one with the highest conditional likelihood,  $\hat{\mathbf{z}} = \operatorname{argmax}_{\mathbf{z}} p(\mathbf{z}^{(1:N_z)} | \mathbf{x}^{(1:N_x)})$ .

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

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

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

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

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

2857 In this case, the sampling distribution does not depend on the other instances  $\mathbf{x}^{(-i)}, \mathbf{z}^{(-i)}$ :  
 2858 given the parameters  $\boldsymbol{\phi}$  and  $\boldsymbol{\mu}$ , the posterior distribution over each  $z^{(i)}$  can be computed  
 2859 from  $\mathbf{x}^{(i)}$  alone.

2860 In sampling algorithms, there are several choices for how to deal with the parameters.  
 2861 One possibility is to sample them too. To do this, we must add them to the generative  
 2862 story, by introducing a prior distribution. For the multinomial and categorical parameters  
 2863 in the EM clustering model, the **Dirichlet distribution** is a typical choice, since it defines  
 2864 a probability on exactly the set of vectors that can be parameters: vectors that sum to one  
 2865 and include only non-negative numbers.<sup>10</sup>

2866 To incorporate this prior, the generative model must augmented to indicate that each  
 2867  $\boldsymbol{\phi}_z \sim \text{Dirichlet}(\boldsymbol{\alpha}_\phi)$ , and  $\boldsymbol{\mu} \sim \text{Dirichlet}(\boldsymbol{\alpha}_\mu)$ . The hyperparameters  $\boldsymbol{\alpha}$  are typically set to

---

<sup>10</sup>If  $\sum_i^K \theta_i = 1$  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.

2868 a constant vector  $\alpha = [\alpha, \alpha, \dots, \alpha]$ . When  $\alpha$  is large, the Dirichlet distribution tends to  
 2869 generate vectors that are nearly uniform; when  $\alpha$  is small, it tends to generate vectors that  
 2870 assign most of their probability mass to a few entries. Given prior distributions over  $\phi$   
 2871 and  $\mu$ , we can now include them in Gibbs sampling, drawing values for these parameters  
 2872 from posterior distributions that are conditioned on the other variables in the model.

2873 Unfortunately, sampling  $\phi$  and  $\mu$  usually leads to slow convergence, meaning that a  
 2874 large number of samples is required before the Markov chain breaks free from the initial  
 2875 conditions. The reason is that the sampling distributions for these parameters are tightly  
 2876 constrained by the cluster memberships  $y^{(i)}$ , which in turn are tightly constrained by the  
 2877 parameters. There are two solutions that are frequently employed:

- 2878 • **Empirical Bayesian** methods maintain  $\phi$  and  $\mu$  as parameters rather than latent  
 2879 variables. They still employ sampling in the E-step of the EM algorithm, but they  
 2880 update the parameters using expected counts that are computed from the samples  
 2881 rather than from parametric distributions. This EM-MCMC hybrid is also known  
 2882 as Monte Carlo Expectation Maximization (MCEM; Wei and Tanner, 1990), and is  
 2883 well-suited for cases in which it is difficult to compute  $q^{(i)}$  directly.
- 2884 • In **collapsed Gibbs sampling**, we analytically integrate  $\phi$  and  $\mu$  out of the model.  
 2885 The cluster memberships  $y^{(i)}$  are the only remaining latent variable; we sample them  
 2886 from the compound distribution,

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

2887 For multinomial and Dirichlet distributions, the sampling distribution can be com-  
 2888 puted in closed form.

2889 MCMC algorithms are guaranteed to converge to the true posterior distribution over  
 2890 the latent variables, but there is no way to know how long this will take. In practice, the  
 2891 rate of convergence depends on initialization, just as expectation-maximization depends  
 2892 on initialization to avoid local optima. Thus, while Gibbs Sampling and other MCMC  
 2893 algorithms provide a powerful and flexible array of techniques for statistical inference in  
 2894 latent variable models, they are not a panacea for the problems experienced by EM.

## 2895 5.5.2 Spectral learning

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

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

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

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

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

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

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

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

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

which is similar to the eigendecomposition  $\mathbf{C} = \mathbf{Q}\Lambda\mathbf{Q}^\top$ . This suggests that simply by finding the eigenvectors and eigenvalues of  $\mathbf{C}$ , we could obtain the parameters  $\phi$  and  $\mu$ , 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  $\mu$  and the columns of  $\Phi$  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  $\phi$  and  $\mu$ , the distribution over clusters can be computed from Bayes' rule:

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

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

2914 **Additional resources**

2915 There are a number of other learning paradigms that deviate from supervised learning.

- 2916     • **Active learning:** the learner selects unlabeled instances and requests annotations (Set-
- 2917         tles, 2012).
- 2918     • **Multiple instance learning:** labels are applied to bags of instances, with a positive
- 2919         label applied if at least one instance in the bag meets the criterion (Dietterich et al.,
- 2920         1997; Maron and Lozano-Pérez, 1998).
- 2921     • **Constraint-driven learning:** supervision is provided in the form of explicit con-
- 2922         straints on the learner (Chang et al., 2007; Ganchev et al., 2010).
- 2923     • **Distant supervision:** noisy labels are generated from an external resource (Mintz
- 2924         et al., 2009, also see § 17.2.3).
- 2925     • **Multitask learning:** the learner induces a representation that can be used to solve
- 2926         multiple classification tasks (Collobert et al., 2011).
- 2927     • **Transfer learning:** the learner must solve a classification task that differs from the
- 2928         labeled data (Pan and Yang, 2010).

2929     Expectation maximization was introduced by Dempster et al. (1977), and is discussed

2930     in more detail by Murphy (2012). Like most machine learning treatments, Murphy focus

2931     on continuous observations and Gaussian likelihoods, rather than the discrete observa-

2932     tions typically encountered in natural language processing. Murphy (2012) also includes

2933     an excellent chapter on MCMC; for a textbook-length treatment, see Robert and Casella

2934     (2013). For still more on Bayesian latent variable models, see Barber (2012), and for ap-

2935     plications of Bayesian models to natural language processing, see Cohen (2016). Surveys

2936     are available for semi-supervised learning (Zhu and Goldberg, 2009) and domain adapta-

2937     tion (Søgaard, 2013), although both pre-date the current wave of interest in deep learning.

2938 **Exercises**

- 2939     1. Derive the expectation maximization update for the parameter  $\mu$  in the EM cluster-
- 2940         ing model.
- 2941     2. The expectation maximization lower bound  $\mathcal{J}$  is defined in Equation 5.10. Prove
- 2942         that the inverse  $-\mathcal{J}$  is convex in  $q$ . You can use the following facts about convexity:

  - 2943         •  $f(\mathbf{x})$  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
  - 2944              $\alpha \in [0, 1]$ .
  - 2945         • 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  $\mathbf{x}$ .

- 2946     •  $\log(x + y) \leq \log x + \log y.$

2947     3. Derive the E-step and M-step updates for the following generative model. You may  
2948       assume that the labels  $y^{(i)}$  are observed, but  $z_m^{(i)}$  is not.

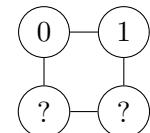
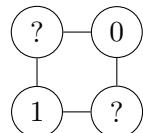
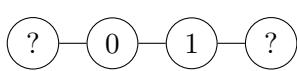
- 2949       • For each instance  $i$ ,

- 2950         – Draw label  $y^{(i)} \sim \text{Categorical}(\boldsymbol{\mu})$
- 2951         – For each token  $m \in \{1, 2, \dots, M^{(i)}\}$ 
  - 2952             \* Draw  $z_m^{(i)} \sim \text{Categorical}(\pi)$
  - 2953             \* If  $z_m^{(i)} = 0$ , draw the current token from a label-specific distribution,  
2954                  $w_m^{(i)} \sim \phi_{y^{(i)}}$
  - 2955             \* If  $z_m^{(i)} = 1$ , draw the current token from a document-specific distribu-  
2956                 tion,  $w_m^{(i)} \sim \nu^{(i)}$

2957     4. Use expectation-maximization clustering to train a word-sense induction system,  
2958       applied to the word *say*.

- 2959       • Import `nltk`, run `nltk.download()` and select `semcor`. Import `semcor`  
2960       from `nltk.corpus`.
- 2961       • The command `semcor.tagged_sentences(tag='sense')` returns an iterator  
2962       over sense-tagged sentences in the corpus. Each sentence can be viewed as  
2963       an iterator over `tree` objects. For `tree` objects that are sense-annotated words,  
2964       you can access the annotation as `tree.label()`, and the word itself with  
2965       `tree.leaves()`. So `semcor.tagged_sentences(tag='sense')[0][2].label()`  
2966       would return the sense annotation of the third word in the first sentence.
- 2967       • Extract all sentences containing the senses `say.v.01` and `say.v.02`.
- 2968       • Build bag-of-words vectors  $x^{(i)}$ , containing the counts of other words in those  
2969       sentences, including all words that occur in at least two sentences.
- 2970       • Implement and run expectation-maximization clustering on the merged data.
- 2971       • Compute the frequency with which each cluster includes instances of `say.v.01`  
2972       and `say.v.02`.

2973     5. Using the iterative updates in Equations 5.34-5.36, compute the outcome of the label  
2974       propagation algorithm for the following examples.



2975     The value inside the node indicates the label,  $y^{(i)} \in \{0, 1\}$ , with  $y^{(i)} = ?$  for unlabeled  
 2976     nodes. The presence of an edge between two nodes indicates  $w_{i,j} = 1$ , and the  
 2977     absence of an edge indicates  $w_{i,j} = 0$ . For the third example, you need only compute  
 2978     the first three iterations, and then you can guess at the solution in the limit.

2979     In the remaining exercises, you will try out some approaches for semisupervised learning  
 2980     and domain adaptation. You will need datasets in multiple domains. You can obtain  
 2981     product reviews in multiple domains here: [https://www.cs.jhu.edu/~mdredze/datasets/sentiment/processed\\_acl.tar.gz](https://www.cs.jhu.edu/~mdredze/datasets/sentiment/processed_acl.tar.gz). Choose a source and target domain,  
 2982     e.g. dvds and books, and divide the data for the target domain into training and test sets  
 2983     of equal size.

- 2985     6. First, quantify the cost of cross-domain transfer.
- 2986         • Train a logistic regression classifier on the source domain training set, and eval-  
 2987         uate it on the target domain test set.
  - 2988         • Train a logistic regression classifier on the target domain training set, and eval-  
 2989         uate it on the target domain test set. This is the “direct transfer” baseline.

2990     Compute the difference in accuracy, which is a measure of the transfer loss across  
 2991     domains.

- 2992     7. Next, apply the **label propagation** algorithm from § 5.3.2.

2993     As a baseline, using only 5% of the target domain training set, train a classifier, and  
 2994     compute its accuracy on the target domain test set.

2995     Next, apply label propagation:

- 2996         • Compute the label matrix  $\mathbf{Q}_L$  for the labeled data (5% of the target domain  
 2997         training set), with each row equal to an indicator vector for the label (positive  
 2998         or negative).
- 2999         • Iterate through the target domain instances, including both test and training  
 3000         data. At each instance  $i$ , compute all  $w_{ij}$ , using Equation 5.32, with  $\alpha = 0.01$ .  
 3001         Use these values to fill in column  $i$  of the transition matrix  $\mathbf{T}$ , setting all but the  
 3002         ten largest values to zero for each column  $i$ . Be sure to normalize the column  
 3003         so that the remaining values sum to one. You may need to use a sparse matrix  
 3004         for this to fit into memory.
- 3005         • Apply the iterative updates from Equations 5.34-5.36 to compute the outcome  
 3006         of the label propagation algorithm for the unlabeled examples.

3007     Select the test set instances from  $\mathbf{Q}_U$ , and compute the accuracy of this method.  
 3008     Compare with the supervised classifier trained only on the 5% sample of the target  
 3009     domain training set.

- 3010     8. Using only 5% of the target domain training data (and all of the source domain train-  
3011       ing data), implement one of the supervised domain adaptation baselines in § 5.4.1.  
3012       See if this improves on the “direct transfer” baseline from the previous problem
- 3013     9. Implement EasyAdapt (§ 5.4.1), again using 5% of the target domain training data  
3014       and all of the source domain data.
- 3015    10. Now try unsupervised domain adaptation, using the “linear projection” method  
3016       described in § 5.4.2. Specifically:
- 3017       • Identify 500 pivot features as the words with the highest frequency in the (com-  
3018          plete) training data for the source and target domains. Specifically, let  $x_i^d$  be the  
3019          count of the word  $i$  in domain  $d$ : choose the 500 words with the largest values  
3020          of  $\min(x_i^{\text{source}}, x_i^{\text{target}})$ .
  - 3021       • Train a classifier to predict each pivot feature from the remaining words in the  
3022          document.
  - 3023       • Arrange the features of these classifiers into a matrix  $\Phi$ , and perform truncated  
3024          singular value decomposition, with  $k = 20$
  - 3025       • Train a classifier from the source domain data, using the combined features  
3026           $\mathbf{x}^{(i)} \oplus \mathbf{U}^\top \mathbf{x}^{(i)}$  — these include the original bag-of-words features, plus the pro-  
3027          jected features.
  - 3028       • Apply this classifier to the target domain test set, and compute the accuracy.



3029

## Part II

3030

# Sequences and trees



# 3031 Chapter 6

## 3032 Language models

3033 In probabilistic classification, the problem is to compute the probability of a label, conditioned on the text. Let's now consider the inverse problem: computing the probability of text itself. Specifically, we will consider models that assign probability to a sequence of word tokens,  $p(w_1, w_2, \dots, w_M)$ , with  $w_m \in \mathcal{V}$ . The set  $\mathcal{V}$  is a discrete vocabulary,

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

3037 Why would you want to compute the probability of a word sequence? In many applications, the goal is to produce word sequences as output:

- 3039 • In **machine translation** (chapter 18), we convert from text in a source language to text in a target language.
- 3040 • In **speech recognition**, we convert from audio signal to text.
- 3041 • In **summarization** (§ 16.3.4.1; § 19.2), we convert from long texts into short texts.
- 3042 • In **dialogue systems** (§ 19.3), we convert from the user's input (and perhaps an external knowledge base) into a text response.

3045 In many of the systems for performing these tasks, there is a subcomponent that computes the probability of the output text. The purpose of this component is to generate texts that are more **fluent**. For example, suppose we want to translate a sentence from Spanish to English.

3049 (6.1) El cafe negro me gusta mucho.

3050 Here is a literal word-for-word translation (a **gloss**):

3051 (6.2) The coffee black me pleases much.

3052 A good language model of English will tell us that the probability of this translation is  
 3053 low, in comparison with more grammatical alternatives,

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

3054 How can we use this fact? Warren Weaver, one of the early leaders in machine trans-  
 3055 lation, viewed it as a problem of breaking a secret code (Weaver, 1955):

3056 When I look at an article in Russian, I say: 'This is really written in English,  
 3057 but it has been coded in some strange symbols. I will now proceed to decode.'

3058 This observation motivates a generative model (like Naïve Bayes):

3059 • The English sentence  $w^{(e)}$  is generated from a **language model**,  $p_e(w^{(e)})$ .

3060 • The Spanish sentence  $w^{(s)}$  is then generated from a **translation model**,  $p_{s|e}(w^{(s)} | w^{(e)})$ .

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

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

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

3061 This is sometimes called the **noisy channel model**, because it envisions English text  
 3062 turning into Spanish by passing through a noisy channel,  $p_{s|e}$ . What is the advantage of  
 3063 modeling translation this way, as opposed to modeling  $p_{e|s}$  directly? The crucial point is  
 3064 that the two distributions  $p_{s|e}$  (the translation model) and  $p_e$  (the language model) can be  
 3065 estimated from separate data. The translation model requires examples of correct trans-  
 3066 lations, but the language model requires only text in English. Such monolingual data is  
 3067 much more widely available. Furthermore, once estimated, the language model  $p_e$  can be  
 3068 reused in any application that involves generating English text, from summarization to  
 3069 speech recognition.

## 3070 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,

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

3071 This estimator is **unbiased**: in the theoretical limit of infinite data, the estimate will  
 3072 be correct. But in practice, we are asking for accurate counts over an infinite number of  
 3073 events, since sequences of words can be arbitrarily long. Even with an aggressive upper  
 3074 bound of, say,  $M = 20$  tokens in the sequence, the number of possible sequences is  $V^{20}$ . A  
 3075 small vocabulary for English would have  $V = 10^4$ , so there are  $10^{80}$  possible sequences.  
 3076 Clearly, this estimator is very data-hungry, and suffers from high variance: even gram-  
 3077 matical sentences will have probability zero if have not occurred in the training data.<sup>1</sup> We  
 3078 therefore need to introduce bias to have a chance of making reliable estimates from finite  
 3079 training data. The language models that follow in this chapter introduce bias in various  
 3080 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,

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

3081 We haven't made any approximations yet, and we could have just as well applied the  
 3082 chain rule in reverse order,

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

3083 or in any other order. But this means that we also haven't really made any progress:  
 3084 to compute the conditional probability  $p(w_M | w_{M-1}, w_{M-2}, \dots, w_1)$ , we would need to  
 3085 model  $V^{M-1}$  contexts. Such a distribution cannot be estimated from any realistic sample  
 3086 of text.

---

<sup>1</sup>Chomsky has famously argued that this is evidence against the very concept of probabilistic language models: no such model could distinguish the grammatical sentence *colorless green ideas sleep furiously* from the ungrammatical permutation *furiously sleep ideas green colorless*. Indeed, even the bigrams in these two examples are unlikely to occur — at least, not in texts written before Chomsky proposed this example.

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

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

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

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

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

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

3087 This model requires estimating and storing the probability of only  $V^n$  events, which is  
 3088 exponential in the order of the  $n$ -gram, and not  $V^M$ , which is exponential in the length of  
 3089 the sentence. The  $n$ -gram probabilities can be computed by relative frequency estimation,

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

3090 The hyperparameter  $n$  controls the size of the context used in each conditional proba-  
 3091 bility. If this is misspecified, the language model will perform poorly. Let's consider the  
 3092 potential problems concretely.

3093 **When  $n$  is too small.** Consider the following sentences:

3094 (6.3) **Gorillas** always like to groom **their** friends.

3095 (6.4) The **computer** that's on the 3rd floor of our office building **crashed**.

3096 In each example, the bolded words depend on each other: the likelihood of *their*  
 3097 depends on knowing that *gorillas* is plural, and the likelihood of *crashed* depends on  
 3098 knowing that the subject is a *computer*. If the  $n$ -grams are not big enough to capture  
 3099 this context, then the resulting language model would offer probabilities that are too  
 3100 low for these sentences, and too high for sentences that fail basic linguistic tests like  
 3101 number agreement.

3102 **When  $n$  is too big.** In this case, it is hard to get good estimates of the  $n$ -gram parameters from  
 3103 our dataset, because of data sparsity. To handle the *gorilla* example, it is necessary to  
 3104 model 6-grams, which means accounting for  $V^6$  events. Under a very small vocab-  
 3105 ularly of  $V = 10^4$ , this means estimating the probability of  $10^{24}$  distinct events.

3106 These two problems point to another **bias-variance tradeoff** (see § 2.1.4). A small  $n$ -  
 3107 gram size introduces high bias, and a large  $n$ -gram size introduces high variance. But  
 3108 in reality we often have both problems at the same time! Language is full of long-range  
 3109 dependencies that we cannot capture because  $n$  is too small; at the same time, language  
 3110 datasets are full of rare phenomena, whose probabilities we fail to estimate accurately  
 3111 because  $n$  is too large. One solution is to try to keep  $n$  large, while still making low-  
 3112 variance estimates of the underlying parameters. To do this, we will introduce a different  
 3113 sort of bias: **smoothing**.

## 3114 6.2 Smoothing and discounting

3115 Limited data is a persistent problem in estimating language models. In § 6.1, we presented  
 3116  $n$ -grams as a partial solution. sparse data can be a problem even for low-order  $n$ -grams;  
 3117 at the same time, many linguistic phenomena, like subject-verb agreement, cannot be in-  
 3118 corporated into language models without high-order  $n$ -grams. It is therefore necessary to  
 3119 add additional inductive biases to  $n$ -gram language models. This section covers some of  
 3120 the most intuitive and common approaches, but there are many more (Chen and Good-  
 3121 man, 1999).

### 3122 6.2.1 Smoothing

3123 A major concern in language modeling is to avoid the situation  $p(w) = 0$ , which could  
 3124 arise as a result of a single unseen n-gram. A similar problem arose in Naïve Bayes, and  
 3125 the solution was **smoothing**: adding imaginary “pseudo” counts. The same idea can be  
 3126 applied to  $n$ -gram language models, as shown here in the bigram case,

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

3127 This basic framework is called **Lidstone smoothing**, but special cases have other names:

- 3128 • **Laplace smoothing** corresponds to the case  $\alpha = 1$ .
- 3129 • **Jeffreys-Perks law** corresponds to the case  $\alpha = 0.5$ . Manning and Schütze (1999)  
 3130 offer more insight on the justifications for this setting.

3131 To maintain normalization, anything that we add to the numerator ( $\alpha$ ) must also ap-  
 3132 pear in the denominator ( $V\alpha$ ). This idea is reflected in the concept of **effective counts**:

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

	counts	unsmoothed probability	Lidstone smoothing, $\alpha = 0.1$		Discounting, $d = 0.1$	
			effective counts	smoothed probability	effective counts	smoothed probability
<i>impropriety</i>	8	0.4	7.826	0.391	7.9	0.395
<i>offense</i>	5	0.25	4.928	0.246	4.9	0.245
<i>damage</i>	4	0.2	3.961	0.198	3.9	0.195
<i>deficiencies</i>	2	0.1	2.029	0.101	1.9	0.095
<i>outbreak</i>	1	0.05	1.063	0.053	0.9	0.045
<i>infirmity</i>	0	0	0.097	0.005	0.25	0.013
<i>cephalopods</i>	0	0	0.097	0.005	0.25	0.013

Table 6.1: Example of Lidstone smoothing and absolute discounting in a bigram language model, for the context *(alleged, -)*, for a toy corpus with a total of twenty counts over the seven words shown. Note that discounting decreases the probability for all but the unseen words, while Lidstone smoothing increases the effective counts and probabilities for *deficiencies* and *outbreak*.

where  $c_i$  is the count of event  $i$ ,  $c_i^*$  is the effective count, and  $M = \sum_{i=1}^V c_i$  is the total number of tokens in the dataset  $(w_1, w_2, \dots, w_M)$ . This term ensures that  $\sum_{i=1}^V c_i^* = \sum_{i=1}^V c_i = M$ . The **discount** for each n-gram is then computed as,

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

### 3133 6.2.2 Discounting and backoff

3134 Discounting “borrows” probability mass from observed  $n$ -grams and redistributes it. In  
 3135 Lidstone smoothing, the borrowing is done by increasing the denominator of the relative  
 3136 frequency estimates. The borrowed probability mass is then redistributed by increasing  
 3137 the numerator for all  $n$ -grams. Another approach would be to borrow the same amount  
 3138 of probability mass from all observed  $n$ -grams, and redistribute it among only the unob-  
 3139 served  $n$ -grams. This is called **absolute discounting**. For example, suppose we set an  
 3140 absolute discount  $d = 0.1$  in a bigram model, and then redistribute this probability mass  
 3141 equally over the unseen words. The resulting probabilities are shown in Table 6.1.

Discounting reserves some probability mass from the observed data, and we need not redistribute this probability mass equally. Instead, we can **backoff** to a lower-order language model: if you have trigrams, use trigrams; if you don’t have trigrams, use bigrams; if you don’t even have bigrams, use unigrams. This is called **Katz backoff**. In the simple

case of backing off from bigrams to unigrams, the bigram probabilities are computed as,

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

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

3142 The term  $\alpha(j)$  indicates the amount of probability mass that has been discounted for  
 3143 context  $j$ . This probability mass is then divided across all the unseen events,  $\{i' : c(i', j) =$   
 3144  $0\}$ , proportional to the unigram probability of each word  $i'$ . The discount parameter  $d$  can  
 3145 be optimized to maximize performance (typically held-out log-likelihood) on a develop-  
 3146 ment set.

### 3147 6.2.3 \*Interpolation

3148 Backoff is one way to combine different order  $n$ -gram models. An alternative approach  
 3149 is **interpolation**: setting the probability of a word in context to a weighted sum of its  
 3150 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,

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

3151 In this equation,  $p_n^*$  is the unsmoothed empirical probability given by an  $n$ -gram lan-  
 3152 guage model, and  $\lambda_n$  is the weight assigned to this model. To ensure that the interpolated  
 3153  $p(w)$  is still a valid probability distribution, the values of  $\lambda$  must obey the constraint,  
 3154  $\sum_{n=1}^{n_{\max}} \lambda_n = 1$ . But how to find the specific values?

3155 An elegant solution is **expectation maximization**. Recall from chapter 5 that we can  
 3156 think about EM as learning with *missing data*: we just need to choose missing data such  
 3157 that learning would be easy if it weren't missing. What's missing in this case? Think of  
 3158 each word  $w_m$  as drawn from an  $n$ -gram of unknown size,  $z_m \in \{1 \dots n_{\max}\}$ . This  $z_m$  is  
 3159 the missing data that we are looking for. Therefore, the application of EM to this problem  
 3160 involves the following **generative process**:

3161 **for** Each token  $w_m, m = 1, 2, \dots, M$  **do**:  
 3162   draw the  $n$ -gram size  $z_m \sim \text{Categorical}(\lambda)$ ;  
 3163   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  $\lambda$  could be estimated as the relative frequency,

$$\lambda_z = \frac{\text{count}(Z_m = z)}{M} \quad [6.17]$$

$$\propto \sum_{m=1}^M \delta(Z_m = z). \quad [6.18]$$

But since we do not know the values of the latent variables  $Z_m$ , we impute a distribution  $q_m$  in the E-step, which represents the degree of belief that word token  $w_m$  was generated from a  $n$ -gram of order  $z_m$ ,

$$q_m(z) \triangleq \Pr(Z_m = z \mid \mathbf{w}_{1:m}; \lambda) \quad [6.19]$$

$$= \frac{p(w_m \mid \mathbf{w}_{1:m-1}, Z_m = z) \times p(z)}{\sum_{z'} p(w_m \mid \mathbf{w}_{1:m-1}, Z_m = z') \times p(z')} \quad [6.20]$$

$$\propto p_z^*(w_m \mid \mathbf{w}_{1:m-1}) \times \lambda_z. \quad [6.21]$$

In the M-step,  $\lambda$  is computed by summing the expected counts under  $q$ ,

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

3165 A solution is obtained by iterating between updates to  $q$  and  $\lambda$ . The complete algorithm  
 3166 is shown in Algorithm 10.

---

**Algorithm 10** Expectation-maximization for interpolated language modeling
 

---

```

1: procedure ESTIMATE INTERPOLATED  $n$ -GRAM ( $\mathbf{w}_{1:M}, \{p_n^*\}_{n \in 1:n_{\max}}$ )
2:   for  $z \in \{1, 2, \dots, n_{\max}\}$  do ▷ Initialization
3:      $\lambda_z \leftarrow \frac{1}{n_{\max}}$ 
4:   repeat
5:     for  $m \in \{1, 2, \dots, M\}$  do ▷ E-step
6:       for  $z \in \{1, 2, \dots, n_{\max}\}$  do
7:          $q_m(z) \leftarrow p_z^*(w_m \mid \mathbf{w}_{1:m-1}) \times \lambda_z$ 
8:        $q_m \leftarrow \text{Normalize}(q_m)$ 
9:     for  $z \in \{1, 2, \dots, n_{\max}\}$  do ▷ M-step
10:     $\lambda_z \leftarrow \frac{1}{M} \sum_{m=1}^M q_m(z)$ 
11:  until tired
12:  return  $\lambda$ 
  
```

---

3167 **6.2.4 \*Kneser-Ney smoothing**

3168 Kneser-Ney smoothing is based on absolute discounting, but it redistributes the result-  
 3169 ing probability mass in a different way from Katz backoff. Empirical evidence points  
 3170 to Kneser-Ney smoothing as the state-of-art for  $n$ -gram language modeling (Goodman,  
 3171 2001). To motivate Kneser-Ney smoothing, consider the example: *I recently visited ..*  
 3172 Which of the following is more likely?

- 3173     • *Francisco*  
 3174     • *Duluth*

3175 Now suppose that both bigrams *visited Duluth* and *visited Francisco* are unobserved in  
 3176 the training data, and furthermore, the unigram probability  $p_1^*(\text{Francisco})$  is greater than  
 3177  $p^*(\text{Duluth})$ . Nonetheless we would still guess that  $p(\text{visited Duluth}) > p(\text{visited Francisco})$ ,  
 3178 because *Duluth* is a more “versatile” word: it can occur in many contexts, while *Francisco*  
 3179 usually occurs in a single context, following the word *San*. This notion of versatility is the  
 3180 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,

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

3181 This is the amount of probability mass left to account for versatility, which we define via  
 3182 the *continuation probability*  $p_{\text{continuation}}(w)$  as proportional to the number of observed con-  
 3183 texts in which  $w$  appears. The numerator of the continuation probability is the number of  
 3184 contexts  $u$  in which  $w$  appears; the denominator normalizes the probability by summing  
 3185 the same quantity over all words  $w'$ .

3186 The idea of modeling versatility by counting contexts may seem heuristic, but there is  
 3187 an elegant theoretical justification from Bayesian nonparametrics (Teh, 2006). Kneser-Ney  
 3188 smoothing on  $n$ -grams was the dominant language modeling technique before the arrival  
 3189 of neural language models.

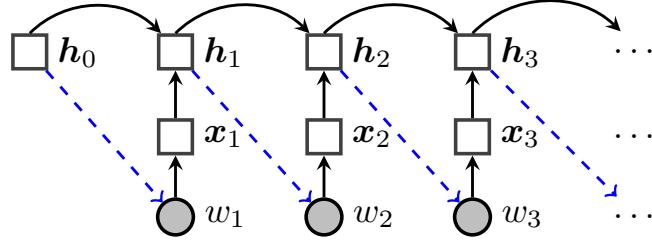


Figure 6.1: The recurrent neural network language model, viewed as an “unrolled” computation graph. Solid lines indicate direct computation, dotted blue lines indicate probabilistic dependencies, circles indicate random variables, and squares indicate computation nodes.

### 3190 6.3 Recurrent neural network language models

3191  $N$ -gram language models have been largely supplanted by **neural networks**. These mod-  
 3192 els do not make the  $n$ -gram assumption of restricted context; indeed, they can incorpo-  
 3193 rate arbitrarily distant contextual information, while remaining computationally and statisti-  
 3194 cally tractable.

3195 The first insight behind neural language models is to treat word prediction as a *dis-  
 3196 criminative* learning task.<sup>2</sup> The goal is to compute the probability  $p(w | u)$ , where  $w \in \mathcal{V}$  is  
 3197 a word, and  $u$  is the context, which depends on the previous words. Rather than directly  
 3198 estimating the word probabilities from (smoothed) relative frequencies, we can treat  
 3199 language modeling as a machine learning problem, and estimate parameters that maxi-  
 3200 mize the log conditional probability of a corpus.

3201 The second insight is to reparametrize the probability distribution  $p(w | u)$  as a func-  
 3202 tion of two dense  $K$ -dimensional numerical vectors,  $\beta_w \in \mathbb{R}^K$ , and  $v_u \in \mathbb{R}^K$ ,

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

3203 where  $\beta_w \cdot v_u$  represents a dot product. As usual, the denominator ensures that the prob-  
 3204 ability distribution is properly normalized. This vector of probabilities is equivalent to  
 3205 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  $\beta_w$  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

---

<sup>2</sup>This idea predates neural language models (e.g., Rosenfeld, 1996; Roark et al., 2007).

but effective neural language model can be built from a **recurrent neural network** (RNN; Mikolov et al., 2010). The basic idea is to recurrently update the context vectors while moving through the sequence. Let  $\mathbf{h}_m$  represent the contextual information at position  $m$  in the sequence. RNN language models are defined,

$$\mathbf{x}_m \triangleq \phi_{w_m} \quad [6.28]$$

$$\mathbf{h}_m = \text{RNN}(\mathbf{x}_m, \mathbf{h}_{m-1}) \quad [6.29]$$

$$p(w_{m+1} | w_1, w_2, \dots, w_m) = \frac{\exp(\beta_{w_{m+1}} \cdot \mathbf{h}_m)}{\sum_{w' \in \mathcal{V}} \exp(\beta_{w'} \cdot \mathbf{h}_m)}, \quad [6.30]$$

where  $\phi$  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).<sup>3</sup> The  $\tanh$  acts as a **squashing function**, ensuring that each element of  $\mathbf{h}_m$  is constrained to the range  $[-1, 1]$ .

Although each  $w_m$  depends on only the context vector  $\mathbf{h}_{m-1}$ , this vector is in turn influenced by *all* previous tokens,  $w_1, w_2, \dots, w_{m-1}$ , through the recurrence operation:  $w_1$  affects  $\mathbf{h}_1$ , which affects  $\mathbf{h}_2$ , and so on, until the information is propagated all the way to  $\mathbf{h}_{m-1}$ , and then on to  $w_m$  (see Figure 6.1). This is an important distinction from  $n$ -gram language models, where any information outside the  $n$ -word window is ignored. In principle, the RNN language model can handle long-range dependencies, such as number agreement over long spans of text — although it would be difficult to know where exactly in the vector  $\mathbf{h}_m$  this information is represented. The main limitation is that information is attenuated by repeated application of the squashing function  $g$ . **Long short-term memories** (LSTMs), described below, are a variant of RNNs that address this issue, using memory cells to propagate information through the sequence without applying nonlinearities (Hochreiter and Schmidhuber, 1997).

The denominator in Equation 6.30 is a computational bottleneck, because it involves a sum over the entire vocabulary. One solution is to use a **hierarchical softmax** function, which computes the sum more efficiently by organizing the vocabulary into a tree (Mikolov et al., 2011). Another strategy is to optimize an alternative metric, such as **noise-contrastive estimation** (Gutmann and Hyvärinen, 2012), which learns by distinguishing observed instances from artificial instances generated from a noise distribution (Mnih and Teh, 2012). Both of these strategies are described in § 14.5.3.

<sup>3</sup>In 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).

3232 **6.3.1 Backpropagation through time**

3233 The recurrent neural network language model has the following parameters:

- 3234 •  $\phi_i \in \mathbb{R}^K$ , the “input” word vectors (these are sometimes called **word embeddings**,  
3235 since each word is embedded in a  $K$ -dimensional space);
- 3236 •  $\beta_i \in \mathbb{R}^K$ , the “output” word vectors;
- 3237 •  $\Theta \in \mathbb{R}^{K \times K}$ , the recurrence operator;
- 3238 •  $\mathbf{h}_0$ , the initial state.

3239 Each of these parameters can be estimated by formulating an objective function over the  
3240 training corpus,  $L(\mathbf{w})$ , and then applying **backpropagation** to obtain gradients on the  
3241 parameters from a minibatch of training examples (see § 3.3.1). Gradient-based updates  
3242 can be computed from an online learning algorithm such as stochastic gradient descent  
3243 (see § 2.5.2).

3244 The application of backpropagation to recurrent neural networks is known as **back-**  
3245 **propagation through time**, because the gradients on units at time  $m$  depend in turn on the  
3246 gradients of units at earlier times  $n < m$ . Let  $\ell_{m+1}$  represent the negative log-likelihood  
3247 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  $\Theta$ . 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  $\Theta$  in several ways. First,  $\mathbf{h}_m$  is computed by multiplying  $\Theta$  by the previous state  $\mathbf{h}_{m-1}$ . But the previous state  $\mathbf{h}_{m-1}$  also depends on  $\Theta$ :

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

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

3248 where  $g'$  is the local derivative of the nonlinear function  $g$ . The key point in this equation  
3249 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  
3250 so on, until reaching the initial state  $\mathbf{h}_0$ .

3251 Each derivative  $\frac{\partial \mathbf{h}_m}{\partial \theta_{k,k'}}$  will be reused many times: it appears in backpropagation from  
3252 the loss  $\ell_m$ , but also in all subsequent losses  $\ell_{n>m}$ . Neural network toolkits such as  
3253 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).<sup>4</sup>

### 6.3.2 Hyperparameters

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

### 6.3.3 Gated recurrent neural networks

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

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

---

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

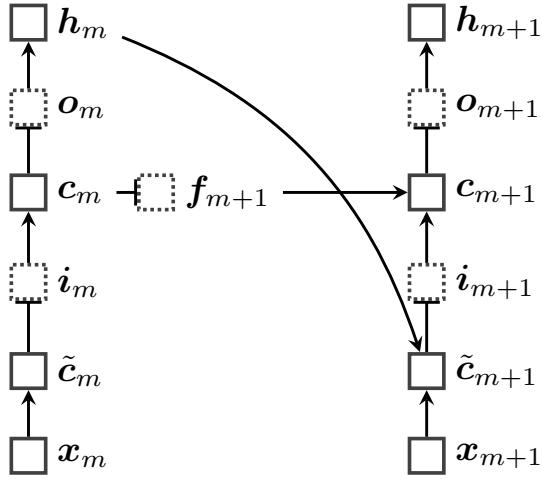


Figure 6.2: The long short-term memory (LSTM) architecture. Gates are shown in boxes with dotted edges. In an LSTM language model, each  $h_m$  would be used to predict the next word  $w_{m+1}$ .

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:

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

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

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

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

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

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

3288 The operator  $\odot$  is an elementwise (Hadamard) product. Each gate is controlled by a vector  
 3289 which parametrize the previous hidden state (e.g.,  $\Theta^{(h \rightarrow f)}$ ) and the current  
 3290 input (e.g.,  $\Theta^{(x \rightarrow f)}$ ), plus a vector offset (e.g.,  $b_f$ ). The overall operation can be infor-  
 3291 mally summarized as  $(h_m, c_m) = \text{LSTM}(x_m, (h_{m-1}, c_{m-1}))$ , with  $(h_m, c_m)$  representing  
 3292 the LSTM state after reading token  $m$ .

3293 The LSTM outperforms standard recurrent neural networks across a wide range of  
 3294 problems. It was first used for language modeling by Sundermeyer et al. (2012), but can  
 3295 be applied more generally: the vector  $h_m$  can be treated as a complete representation of

3296 the input sequence up to position  $m$ , and can be used for any labeling task on a sequence  
 3297 of tokens, as we will see in the next chapter.

3298 There are several LSTM variants, of which the Gated Recurrent Unit (Cho et al., 2014)  
 3299 is one of the more well known. Many software packages implement a variety of RNN  
 3300 architectures, so choosing between them is simple from a user’s perspective. Jozefowicz  
 3301 et al. (2015) provide an empirical comparison of various modeling choices circa 2015.

## 3302 6.4 Evaluating language models

3303 Language modeling is not usually an application in itself: language models are typically  
 3304 components of larger systems, and they would ideally be evaluated **extrinsically**. This  
 3305 means evaluating whether the language model improves performance on the application  
 3306 task, such as machine translation or speech recognition. But this is often hard to do, and  
 3307 depends on details of the overall system which may be irrelevant to language modeling.  
 3308 In contrast, **intrinsic evaluation** is task-neutral. Better performance on intrinsic metrics  
 3309 may be expected to improve extrinsic metrics across a variety of tasks, but there is always  
 3310 the risk of over-optimizing the intrinsic metric. This section discusses some intrinsic met-  
 3311 rics, but keep in mind the importance of performing extrinsic evaluations to ensure that  
 3312 intrinsic performance gains carry over to the applications that we care about.

### 3313 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,

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

3314 treating the entire held-out corpus as a single stream of tokens.

3315 Typically, unknown words are mapped to the  $\langle \text{UNK} \rangle$  token. This means that we have  
 3316 to estimate some probability for  $\langle \text{UNK} \rangle$  on the training data. One way to do this is to fix  
 3317 the vocabulary  $\mathcal{V}$  to the  $V - 1$  words with the highest counts in the training data, and then  
 3318 convert all other tokens to  $\langle \text{UNK} \rangle$ . Other strategies for dealing with out-of-vocabulary  
 3319 terms are discussed in § 6.5.

3320 **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,

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

3321 where  $M$  is the total number of tokens in the held-out corpus.

3322 Lower perplexities correspond to higher likelihoods, so lower scores are better on this  
3323 metric — it is better to be less perplexed. Here are some special cases:

- 3324 • In the limit of a perfect language model, probability 1 is assigned to the held-out  
3325 corpus, with  $\text{Perplex}(\mathbf{w}) = 2^{-\frac{1}{M} \log_2 1} = 2^0 = 1$ .
- 3326 • In the opposite limit, probability zero is assigned to the held-out corpus, which cor-  
3327 responds to an infinite perplexity,  $\text{Perplex}(\mathbf{w}) = 2^{-\frac{1}{M} \log_2 0} = 2^\infty = \infty$ .
- 3328 • Assume a uniform, unigram model in which  $p(w_i) = \frac{1}{V}$  for all words in the vocab-  
3329 ular. Then,

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

3328 This is the “worst reasonable case” scenario, since you could build such a language  
3329 model without even looking at the data.

3330 In practice, language models tend to give perplexities in the range between 1 and  $V$ .  
3331 A small benchmark dataset is the **Penn Treebank**, which contains roughly a million to-  
3332 kens; its vocabulary is limited to 10,000 words, with all other tokens mapped a special  
3333  $\langle \text{UNK} \rangle$  symbol. On this dataset, a well-smoothed 5-gram model achieves a perplexity of  
3334 141 (Mikolov and Zweig, Mikolov and Zweig), and an LSTM language model achieves  
3335 perplexity of roughly 80 (Zaremba, Sutskever, and Vinyals, Zaremba et al.). Various en-  
3336 hancements to the LSTM architecture can bring the perplexity below 60 (Merity et al.,  
3337 2018). A larger-scale language modeling dataset is the 1B Word Benchmark (Chelba et al.,  
3338 2013), which contains text from Wikipedia. On this dataset, a perplexities of around 25  
3339 can be obtained by averaging together multiple LSTM language models (Jozefowicz et al.,  
3340 2016).

3341 **6.5 Out-of-vocabulary words**

3342 So far, we have assumed a **closed-vocabulary** setting — the vocabulary  $\mathcal{V}$  is assumed to be  
 3343 a finite set. In realistic application scenarios, this assumption may not hold. Consider, for  
 3344 example, the problem of translating newspaper articles. The following sentence appeared  
 3345 in a Reuters article on January 6, 2017:<sup>5</sup>

3346 The report said U.S. intelligence agencies believe Russian military intelligence,  
 3347 the **GRU**, used intermediaries such as **WikiLeaks**, **DCLeaks.com** and the **Guc-**  
 3348 **cifer** 2.0 "persona" to release emails...

3349 Suppose that you trained a language model on the Gigaword corpus,<sup>6</sup> which was released  
 3350 in 2003. The bolded terms either did not exist at this date, or were not widely known; they  
 3351 are unlikely to be in the vocabulary. The same problem can occur for a variety of other  
 3352 terms: new technologies, previously unknown individuals, new words (e.g., *hashtag*), and  
 3353 numbers.

3354 One solution is to simply mark all such terms with a special token,  $\langle \text{UNK} \rangle$ . While  
 3355 training the language model, we decide in advance on the vocabulary (often the  $K$  most  
 3356 common terms), and mark all other terms in the training data as  $\langle \text{UNK} \rangle$ . If we do not want  
 3357 to determine the vocabulary size in advance, an alternative approach is to simply mark  
 3358 the first occurrence of each word type as  $\langle \text{UNK} \rangle$ .

3359 But it often better to make distinctions about the likelihood of various unknown words.  
 3360 This is particularly important in languages that have rich morphological systems, with  
 3361 many inflections for each word. For example, Portuguese is only moderately complex  
 3362 from a morphological perspective, yet each verb has dozens of inflected forms (see Fig-  
 3363 ure 4.3b). In such languages, there will be many word types that we do not encounter in a  
 3364 corpus, which are nonetheless predictable from the morphological rules of the language.  
 3365 To use a somewhat contrived English example, if *transfenestrate* is in the vocabulary, our  
 3366 language model should assign a non-zero probability to the past tense *transfenestrated*,  
 3367 even if it does not appear in the training data.

3368 One way to accomplish this is to supplement word-level language models with **character-**  
 3369 **level language models**. Such models can use  $n$ -grams or RNNs, but with a fixed vocab-  
 3370 uary equal to the set of ASCII or Unicode characters. For example Ling et al. (2015)  
 3371 propose an LSTM model over characters, and Kim (2014) employ a **convolutional neural**  
 3372 **network** (LeCun and Bengio, 1995). A more linguistically motivated approach is to seg-  
 3373 ment words into meaningful subword units, known as **morphemes** (see chapter 9). For

---

<sup>5</sup>Bayoumy, Y. and Strobel, W. (2017, January 6). U.S. intel report: Putin directed cy-  
 ber campaign to help Trump. *Reuters*. Retrieved from <http://www.reuters.com/article/us-usa-russia-cyber-idUSKBN14Q1T8> on January 7, 2017.

<sup>6</sup><https://catalog.ldc.upenn.edu/LDC2003T05>

3374 example, Botha and Blunsom (2014) induce vector representations for morphemes, which  
 3375 they build into a log-bilinear language model; Bhatia et al. (2016) incorporate morpheme  
 3376 vectors into an LSTM.

## 3377 Additional resources

3378 A variety of neural network architectures have been applied to language modeling. No-  
 3379 table earlier non-recurrent architectures include the neural probabilistic language model (Ben-  
 3380 gio et al., 2003) and the log-bilinear language model (Mnih and Hinton, 2007). Much more  
 3381 detail on these models can be found in the text by Goodfellow et al. (2016).

## 3382 Exercises

- 3383 1. Prove that  $n$ -gram language models give valid probabilities if the  $n$ -gram probabili-  
 3384 ties are valid. Specifically, assume that,

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

3385 for all contexts  $(w_{m-1}, w_{m-2}, \dots, w_{m-n+1})$ . Prove that  $\sum_w p_n(w) = 1$  for all  $w \in \mathcal{V}^*$ ,  
 3386 where  $p_n$  is the probability under an  $n$ -gram language model. Your proof should  
 3387 proceed by induction. You should handle the start-of-string case  $p(w_1 | \underbrace{\square, \dots, \square}_{n-1})$ ,  
 3388 but you need not handle the end-of-string token.

- 3389 2. First, show that RNN language models are valid using a similar proof technique to  
 3390 the one in the previous problem.

3391 Next, let  $p_r(w)$  indicate the probability of  $w$  under RNN  $r$ . An ensemble of RNN  
 3392 language models computes the probability,

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

3393 Does an ensemble of RNN language models compute a valid probability?

- 3394 3. Consider a unigram language model over a vocabulary of size  $V$ . Suppose that a  
 3395 word appears  $m$  times in a corpus with  $M$  tokens in total. With Lidstone smoothing  
 3396 of  $\alpha$ , for what values of  $m$  is the smoothed probability greater than the unsmoothed  
 3397 probability?

- 3398 4. Consider a simple language in which each token is drawn from the vocabulary  $\mathcal{V}$   
 3399 with probability  $\frac{1}{V}$ , independent of all other tokens.

3400 Given a corpus of size  $M$ , what is the expectation of the fraction of all possible  
 3401 bigrams that have zero count? You may assume  $V$  is large enough that  $\frac{1}{V} \approx \frac{1}{V-1}$ .

- 3402 5. Continuing the previous problem, determine the value of  $M$  such that the fraction  
 3403 of bigrams with zero count is at most  $\epsilon \in (0, 1)$ . As a hint, you may use the approxi-  
 3404 mation  $\ln(1 + \alpha) \approx \alpha$  for  $\alpha \approx 0$ .

- 3405 6. In real languages, words probabilities are neither uniform nor independent.

- 3406 • Assume that word probabilities are independent but not uniform, so that in  
 3407 general  $p(w) \neq \frac{1}{V}$ . Prove that the expected fraction of unseen bigrams will be  
 3408 higher than in the IID case.
- 3409 • Assume that word probabilities are neither independent nor uniform. Again,  
 3410 prove that the expected fraction of unseen bigrams will be higher than in the  
 3411 IID case. [todo: double check]

- 3412 7. Consider a recurrent neural network with a single hidden unit and a sigmoid acti-  
 3413 vation,  $h_m = \sigma(\theta h_{m-1} + x_m)$ . Prove that if  $|\theta| < 1$ , then the gradient  $\frac{\partial h_m}{\partial h_{m-k}}$  goes to  
 3414 zero as  $k \rightarrow \infty$ .<sup>7</sup>

- 3415 8. **Zipf's law** states that if the word types in a corpus are sorted by frequency, then the  
 3416 frequency of the word at rank  $r$  is proportional to  $r^{-s}$ , where  $s$  is a free parameter,  
 3417 usually around 1. (Another way to view Zipf's law is that a plot of log frequency  
 3418 against log rank will be linear.) Solve for  $s$  using the counts of the first and second  
 3419 most frequent words,  $c_1$  and  $c_2$ .

- 3420 9. Download the wikitext-2 dataset.<sup>8</sup> Read and tokenize the training data, e.g. with the  
 3421 CountVectorizer class from scikit-learn. Estimate the Zipf's law coefficient  
 3422 by,

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

3423 where  $\mathbf{r} = [1, 2, 3, \dots]$  is the vector of ranks of all words in the corpus, and  $\mathbf{c} =$   
 3424  $[c_1, c_2, c_3, \dots]$  is the vector of counts of all words in the corpus, sorted in descending  
 3425 order.

<sup>7</sup>This proof generalizes to vector hidden units by considering the largest eigenvector of the matrix  $\Theta$  (Pascanu et al., 2013).

<sup>8</sup>Currently available at [https://github.com/pytorch/examples/tree/master/word\\_language\\_model/data/wikitext-2](https://github.com/pytorch/examples/tree/master/word_language_model/data/wikitext-2)

- 3426        Make a log-log plot of the observed counts, and the expected counts according to  
3427        Zipf's law. The sum  $\sum_{r=1}^{\infty} r^s = \zeta(s)$  is the Riemann zeta function, available in  
3428        python's `scipy` library as `scipy.special.zeta`.
- 3429        10. Using the Pytorch library, train an LSTM language model from the Wikitext train-  
3430        ing corpus. After each epoch of training, compute its perplexity on the Wikitext  
3431        validation corpus. For this exercise, you can focus on the  $10^4$  most frequent words  
3432        in the training corpus, and label everything else as `<UNK>`. Stop training when the  
3433        perplexity stops improving.

## 3434 Chapter 7

# 3435 Sequence labeling

3436 The goal of sequence labeling is to assign tags to words, or more generally, to assign discrete labels to discrete elements in a sequence. There are many applications of sequence labeling in natural language processing, and chapter 8 presents an overview. A classic application is **part-of-speech tagging**, which involves tagging each word by its grammatical category. Coarse-grained grammatical categories include **NOUNs**, which describe things, properties, or ideas, and **VERBs**, which describe actions and events. Consider a simple input:

3443 (7.1) They can fish.

3444 A dictionary of coarse-grained part-of-speech tags might include **NOUN** as the only valid tag for *they*, but both **NOUN** and **VERB** as potential tags for *can* and *fish*. An accurate sequence labeling algorithm should select the verb tag for both *can* and *fish* in (7.1), but it should select the noun tags for the same two words in the phrase *can of fish*.

### 3448 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:

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

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

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

3449 Here the feature function takes three arguments as input: the sentence to be tagged (e.g., *they can fish*), the proposed tag (e.g., N or V), and the index of the token to which this tag

3451 is applied. This simple feature function then returns a single feature: a tuple including  
 3452 the word to be tagged and the tag that has been proposed. If the vocabulary size is  $V$   
 3453 and the number of tags is  $K$ , then there are  $V \times K$  features. Each of these features must  
 3454 be assigned a weight. These weights can be learned from a labeled dataset using a clas-  
 3455 sification algorithm such as perceptron, but this isn't necessary in this case: it would be  
 3456 equivalent to define the classification weights directly, with  $\theta_{w,y} = 1$  for the tag  $y$  most  
 3457 frequently associated with word  $w$ , and  $\theta_{w,y} = 0$  for all other tags.

However, it is easy to see that this simple classification approach cannot correctly tag both *they can fish* and *can of fish*, because *can* and *fish* are grammatically ambiguous. To handle both of these cases, the tagger must rely on context, such as the surrounding words. We can build context into the feature set by incorporating the surrounding words as additional features:

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

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

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

3458 These features contain enough information that a tagger should be able to choose the  
 3459 right tag for the word *fish*: words that come after *can* are likely to be verbs, so the feature  
 3460  $(w_{m-1} = \text{can}, y_m = \mathbf{V})$  should have a large positive weight.

3461 However, even with this enhanced feature set, it may be difficult to tag some se-  
 3462 quences correctly. One reason is that there are often relationships between the tags them-  
 3463 selves. For example, in English it is relatively rare for a verb to follow another verb —  
 3464 particularly if we differentiate MODAL verbs like *can* and *should* from more typical verbs,  
 3465 like *give*, *transcend*, and *befuddle*. We would like to incorporate preferences against tag se-  
 3466 quences like VERB-VERB, and in favor of tag sequences like NOUN-VERB. The need for  
 3467 such preferences is best illustrated by a **garden path sentence**:

3468 (7.2) The old man the boat.

3469 Grammatically, the word *the* is a DETERMINER. When you read the sentence, what  
 3470 part of speech did you first assign to *old*? Typically, this word is an ADJECTIVE — abbrevi-  
 3471 ated as J — which is a class of words that modify nouns. Similarly, *man* is usually a noun.  
 3472 The resulting sequence of tags is D J N D N. But this is a mistaken “garden path” inter-  
 3473 pretation, which ends up leading nowhere. It is unlikely that a determiner would directly

3474 follow a noun,<sup>1</sup> and it is particularly unlikely that the entire sentence would lack a verb.  
 3475 The only possible verb in (7.2) is the word *man*, which can refer to the act of maintaining  
 3476 and piloting something — often boats. But if *man* is tagged as a verb, then *old* is seated  
 3477 between a determiner and a verb, and must be a noun. And indeed, adjectives often have  
 3478 a second interpretation as nouns when used in this way (e.g., *the young*, *the restless*). This  
 3479 reasoning, in which the labeling decisions are intertwined, cannot be applied in a setting  
 3480 where each tag is produced by an independent classification decision.

## 3481 7.2 Sequence labeling as structure prediction

3482 As an alternative, think of the entire sequence of tags as a label itself. For a given sequence  
 3483 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  
 3484  $\mathcal{Y} = \{\text{N, V, D, ...}\}$  refers to the set of individual tags, and  $\mathcal{Y}^M$  refers to the set of tag  
 3485 sequences of length  $M$ . We can then treat the sequence labeling problem as a classification  
 3486 problem in the label space  $\mathcal{Y}(\mathbf{w})$ ,

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}(\mathbf{w})}{\operatorname{argmax}} \Psi(\mathbf{w}, \mathbf{y}), \quad [7.7]$$

3487 where  $\mathbf{y} = (y_1, y_2, \dots, y_M)$  is a sequence of  $M$  tags, and  $\Psi$  is a scoring function on pairs  
 3488 of sequences,  $V^M \times \mathcal{Y}^M \mapsto \mathbb{R}$ . Such a function can include features that capture the rela-  
 3489 tionships between tagging decisions, such as the preference that determiners not follow  
 3490 nouns, or that all sentences have verbs.

3491 Given that the label space is exponentially large in the length of the sequence  $M$ , can  
 3492 it ever be practical to perform tagging in this way? The problem of making a series of in-  
 3493 terconnected labeling decisions is known as **inference**. Because natural language is full of  
 3494 interrelated grammatical structures, inference is a crucial aspect of natural language pro-  
 3495 cessing. In English, it is not unusual to have sentences of length  $M = 20$ ; part-of-speech  
 3496 tag sets vary in size from 10 to several hundred. Taking the low end of this range, we have  
 3497  $|\mathcal{Y}(\mathbf{w}_{1:M})| \approx 10^{20}$ , one hundred billion billion possible tag sequences. Enumerating and  
 3498 scoring each of these sequences would require an amount of work that is exponential in  
 3499 the sequence length, so inference is intractable.

However, the situation changes when we restrict the scoring function. Suppose we choose a function that decomposes into a sum of local parts,

$$\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m), \quad [7.8]$$

3500 where each  $\psi(\cdot)$  scores a local part of the tag sequence. Note that the sum goes up to  $M+1$ ,  
 3501 so that we can include a score for a special end-of-sequence tag,  $\psi(\mathbf{w}_{1:M}, \diamond, y_M, M+1)$ .  
 3502 We also define a special tag to begin the sequence,  $y_0 \triangleq \diamond$ .

---

<sup>1</sup>The main exception occurs with ditransitive verbs, such as *They gave the winner a trophy*.

3503 In a linear model, local scoring function can be defined as a dot product of weights  
 3504 and features,

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

3505 The feature vector  $\mathbf{f}$  can consider the entire input  $\mathbf{w}$ , and can look at pairs of adjacent  
 3506 tags. This is a step up from per-token classification: the weights can assign low scores  
 3507 to infelicitous tag pairs, such as noun-determiner, and high scores for frequent tag pairs,  
 3508 such as determiner-noun and noun-verb.

In the example *they can fish*, a minimal feature function would include features for word-tag pairs (sometimes called **emission features**) and tag-tag pairs (sometimes called **transition features**):

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

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

3509 There are seven active features for this example: one for each word-tag pair, and one  
 3510 for each tag-tag pair, including a final tag  $y_{M+1} = \blacklozenge$ . These features capture the two main  
 3511 sources of information for part-of-speech tagging in English: which tags are appropriate  
 3512 for each word, and which tags tend to follow each other in sequence. Given appropriate  
 3513 weights for these features, taggers can achieve high accuracy, even for difficult cases like  
 3514 *the old man the boat*. We will now discuss how this restricted scoring function enables  
 3515 efficient inference, through the **Viterbi algorithm** (Viterbi, 1967).

3516 **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:

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} \Psi(\mathbf{w}, \mathbf{y}) \quad [7.13]$$

$$= \operatorname{argmax}_{\mathbf{y}_{1:M}} \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.14]$$

$$= \operatorname{argmax}_{\mathbf{y}_{1:M}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}), \quad [7.15]$$

3517 where the final line simplifies the notation with the shorthand,

$$s_m(y_m, y_{m-1}) \triangleq \psi(\mathbf{w}_{1:M}, y_m, y_{m-1}, m). \quad [7.16]$$

This inference problem can be solved efficiently using **dynamic programming**, a algorithmic technique for reusing work in recurrent computations. As is often the case in dynamic programming, we begin by solving an auxiliary problem: rather than finding the best tag sequence, we simply compute the *score* of the best tag sequence,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}, \mathbf{y}_{1:M}) = \max_{\mathbf{y}_{1:M}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}). \quad [7.17]$$

This score involves a maximization over all tag sequences of length  $M$ , written  $\max_{\mathbf{y}_{1:M}}$ . This maximization can be broken into two pieces,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}, \mathbf{y}_{1:M}) = \max_{y_M} \max_{\mathbf{y}_{1:M-1}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}), \quad [7.18]$$

which simply says that we maximize over the final tag  $y_M$ , and we maximize over all “prefixes”,  $\mathbf{y}_{1:M-1}$ . But within the sum of scores, only the final term  $s_{M+1}(\blacklozenge, y_M)$  depends on  $y_M$ . We can pull this term out of the second maximization,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}, \mathbf{y}_{1:M}) = \max_{y_M} s_{M+1}(\blacklozenge, y_M) + \max_{\mathbf{y}_{1:M-1}} \sum_{m=1}^M s_m(y_m, y_{m-1}). \quad [7.19]$$

This same reasoning can be applied recursively to the second term of Equation 7.19, pulling out  $s_M(y_M, y_{M-1})$ , and so on. We can formalize this idea by defining an auxiliary

---

**Algorithm 11** The Viterbi algorithm. Each  $s_m(k, k')$  is a local score for tag  $y_m = k$  and  $y_{m-1} = k'$ .

---

```

for  $k \in \{0, \dots, K\}$  do
     $v_1(k) = s_1(k, \diamond)$ 
for  $m \in \{2, \dots, M\}$  do
    for  $k \in \{0, \dots, K\}$  do
         $v_m(k) = \max_{k'} s_m(k, k') + v_{m-1}(k')$ 
         $b_m(k) = \operatorname{argmax}_{k'} s_m(k, k') + v_{m-1}(k')$ 
     $y_M = \operatorname{argmax}_k s_{M+1}(\blacklozenge, k) + v_M(k)$ 
    for  $m \in \{M-1, \dots, 1\}$  do
         $y_m = b_m(y_{m+1})$ 
return  $\mathbf{y}_{1:M}$ 
```

---

**Viterbi variable,**

$$v_m(y_m) \triangleq \max_{\mathbf{y}_{1:m-1}} \sum_{n=1}^m s_n(y_n, y_{n-1}) \quad [7.20]$$

$$= \max_{y_{m-1}} s_m(y_m, y_{m-1}) + \max_{\mathbf{y}_{1:m-2}} \sum_{n=1}^{m-1} s_n(y_n, y_{n-1}) \quad [7.21]$$

$$= \max_{y_{m-1}} s_m(y_m, y_{m-1}) + v_{m-1}(y_{m-1}). \quad [7.22]$$

3518 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,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}_{1:M}, \mathbf{y}_{1:M}) = v_{M+1}(\blacklozenge). \quad [7.24]$$

3519 Thus, the score of the best labeling for the sequence can be computed in a single forward  
 3520 sweep: first compute all variables  $v_1(\cdot)$  from Equation 7.23, and then compute all variables  
 3521  $v_2(\cdot)$  from the recurrence Equation 7.22, and continue until reaching the final variable  
 3522  $v_{M+1}(\blacklozenge)$ .

3523 Graphically, it is customary to arrange these variables in a structure known as a **trellis**,  
 3524 shown in Figure 7.1. Each column indexes a token  $m$  in the sequence, and each row

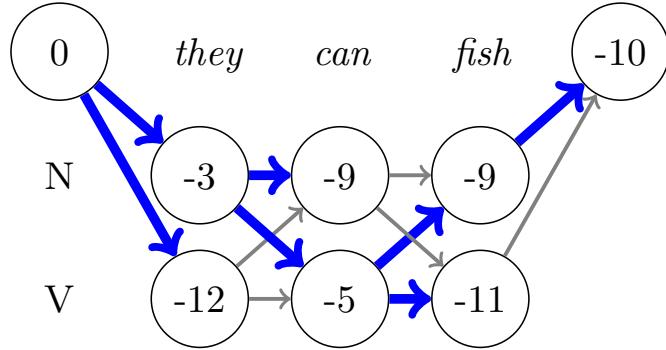


Figure 7.1: The trellis representation of the Viterbi variables, for the example *they can fish*, using the weights shown in Table 7.1.

3525 indexes a tag in  $\mathcal{Y}$ ; every  $v_{m-1}(k)$  is connected to every  $v_m(k')$ , that  $v_m(k')$  is computed  
 3526 from  $v_{m-1}(k)$ . Special nodes are set aside for the start and end states.

3527 Our real goal is to find the best scoring sequence, not simply to compute its score.  
 3528 But solving the auxiliary problem gets us almost all the way there. Recall that each  $v_m(k)$   
 3529 represents the score of the best tag sequence ending in that tag  $k$  in position  $m$ . To compute  
 3530 this, we maximize over possible values of  $y_{m-1}$ . If we keep track of the “argmax” tag that  
 3531 maximizes this choice at each step, then we can walk backwards from the final tag, and  
 3532 recover the optimal tag sequence. This is indicated in Figure 7.1 by the solid blue lines,  
 3533 which we trace back from the final position. These “back-pointers” are written  $b_m(k)$ ,  
 3534 indicating the optimal tag  $y_{m-1}$  on the path to  $Y_m = k$ .

3535 The complete Viterbi algorithm is shown in Algorithm 11. When computing the initial  
 3536 Viterbi variables  $v_1(\cdot)$ , we use a special tag,  $\diamond$ , to indicate the start of the sequence. When  
 3537 computing the final tag  $Y_M$ , we use another special tag,  $\blacklozenge$ , to indicate the end of the  
 3538 sequence. Linguistically, these special tags enable the use of transition features for the tags  
 3539 that begin and end the sequence: for example, conjunctions are unlikely to end sentences  
 3540 in English, so we would like a low score for  $s_{M+1}(\blacklozenge, CC)$ ; nouns are relatively likely to  
 3541 appear at the beginning of sentences, so we would like a high score for  $s_1(N, \diamond)$ , assuming  
 3542 the noun tag is compatible with the first word token  $w_1$ .

3543 **Complexity** If there are  $K$  tags and  $M$  positions in the sequence, then there are  $M \times K$   
 3544 Viterbi variables to compute. Computing each variable requires finding a maximum over  
 3545  $K$  possible predecessor tags. The total time complexity of populating the trellis is therefore  
 3546  $\mathcal{O}(MK^2)$ , with an additional factor for the number of active features at each position.  
 3547 After completing the trellis, we simply trace the backwards pointers to the beginning of  
 3548 the sequence, which takes  $\mathcal{O}(M)$  operations.

	<i>they</i>	<i>can</i>	<i>fish</i>	
N	-2	-3	-3	
V	-10	-1	-3	

(a) Weights for emission features.

	N	V	♦
◊	-1	-2	$-\infty$
N	-3	-1	-1
V	-1	-3	-1

(b) Weights for transition features. The “from” tags are on the columns, and the “to” tags are on the rows.

Table 7.1: Feature weights for the example trellis shown in Figure 7.1. Emission weights from  $\diamond$  and ♦ are implicitly set to  $-\infty$ .3549 

### 7.3.1 Example

3550 Consider the minimal tagset  $\{N, V\}$ , corresponding to nouns and verbs. Even in this  
 3551 tagset, there is considerable ambiguity: for example, the words *can* and *fish* can each take  
 3552 both tags. Of the  $2 \times 2 \times 2 = 8$  possible taggings for the sentence *they can fish*, four are  
 3553 possible given these possible tags, and two are grammatical.<sup>2</sup>

3554 The values in the trellis in Figure 7.1 are computed from the feature weights defined in  
 3555 Table 7.1. We begin with  $v_1(N)$ , which has only one possible predecessor, the start tag  $\diamond$ .  
 3556 This score is therefore equal to  $s_1(N, \diamond) = -2 - 1 = -3$ , which is the sum of the scores for  
 3557 the emission and transition features respectively; the backpointer is  $b_1(N) = \diamond$ . The score  
 3558 for  $v_1(V)$  is computed in the same way:  $s_1(V, \diamond) = -10 - 2 = -12$ , and again  $b_1(V) = \diamond$ .  
 3559 The backpointers are represented in the figure by thick lines.

Things get more interesting at  $m = 2$ . The score  $v_2(N)$  is computed by maximizing over the two possible predecessors,

$$v_2(N) = \max(v_1(N) + s_2(N, N), v_1(V) + s_2(N, V)) \quad [7.25]$$

$$= \max(-3 - 3 - 3, -12 - 3 - 1) = -9 \quad [7.26]$$

$$b_2(N) = N. \quad [7.27]$$

This continues until reaching  $v_4(\diamond)$ , which is computed as,

$$v_4(\diamond) = \max(v_3(N) + s_4(\diamond, N), v_3(V) + s_4(\diamond, V)) \quad [7.28]$$

$$= \max(-9 + 0 - 1, -11 + 0 - 1) \quad [7.29]$$

$$= -10, \quad [7.30]$$

3560 so  $b_4(\diamond) = N$ . As there is no emission  $w_4$ , the emission features have scores of zero.

---

<sup>2</sup>The tagging *they/N can/V fish/N* corresponds to the scenario of putting fish into cans, or perhaps of firing them.

3561 To compute the optimal tag sequence, we walk backwards from here, next checking  
 3562  $b_3(N) = V$ , and then  $b_2(V) = N$ , and finally  $b_1(N) = \diamond$ . This yields  $y = (N, V, N)$ , which  
 3563 corresponds to the linguistic interpretation of the fishes being put into cans.

3564 **7.3.2 Higher-order features**

3565 The Viterbi algorithm was made possible by a restriction of the scoring function to local  
 3566 parts that consider only pairs of adjacent tags. We can think of this as a bigram language  
 3567 model over tags. A natural question is how to generalize Viterbi to tag trigrams, which  
 3568 would involve the following decomposition:

$$\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+2} f(\mathbf{w}, y_m, y_{m-1}, y_{m-2}, m), \quad [7.31]$$

3569 where  $y_{-1} = \diamond$  and  $y_{M+2} = \blacklozenge$ .

3570 One solution is to create a new tagset  $\mathcal{Y}^{(2)}$  from the Cartesian product of the original  
 3571 tagset with itself,  $\mathcal{Y}^{(2)} = \mathcal{Y} \times \mathcal{Y}$ . The tags in this product space are ordered pairs, rep-  
 3572 resenting adjacent tags at the token level: for example, the tag  $(N, V)$  would represent a  
 3573 noun followed by a verb. Transitions between such tags must be consistent: we can have a  
 3574 transition from  $(N, V)$  to  $(V, N)$  (corresponding to the tag sequence  $N V N$ ), but not from  
 3575  $(N, V)$  to  $(N, N)$ , which would not correspond to any coherent tag sequence. This con-  
 3576 straint can be enforced in feature weights, with  $\theta_{((a,b),(c,d))} = -\infty$  if  $b \neq c$ . The remaining  
 3577 feature weights can encode preferences for and against various tag trigrams.

3578 In the Cartesian product tag space, there are  $K^2$  tags, suggesting that the time com-  
 3579 plexity will increase to  $\mathcal{O}(MK^4)$ . However, it is unnecessary to max over predecessor tag  
 3580 bigrams that are incompatible with the current tag bigram. By exploiting this constraint,  
 3581 it is possible to limit the time complexity to  $\mathcal{O}(MK^3)$ . The space complexity grows to  
 3582  $\mathcal{O}(MK^2)$ , since the trellis must store all possible predecessors of each tag. In general, the  
 3583 time and space complexity of higher-order Viterbi grows exponentially with the order of  
 3584 the tag  $n$ -grams that are considered in the feature decomposition.

3585 **7.4 Hidden Markov Models**

3586 Let us now consider how to learn the scores  $s_m(y, y')$  that parametrize the Viterbi sequence  
 3587 labeling algorithm, beginning with a probabilistic approach. Recall from § 2.1 that the  
 3588 probabilistic Naïve Bayes classifier selects the label  $y$  to maximize  $p(y | \mathbf{x}) \propto p(y, \mathbf{x})$ . In  
 3589 probabilistic sequence labeling, our goal is similar: select the tag sequence that maximizes  
 3590  $p(y | \mathbf{w}) \propto p(y, \mathbf{w})$ . The locality restriction in Equation 7.8 can be viewed as a conditional  
 3591 independence assumption on the random variables  $y$ .

**Algorithm 12** Generative process for the hidden Markov model

---

```

 $y_0 \leftarrow \diamond,$     $m \leftarrow 1$ 
repeat
     $y_m \sim \text{Categorical}(\lambda_{y_{m-1}})$             $\triangleright$  sample the current tag
     $w_m \sim \text{Categorical}(\phi_{y_m})$             $\triangleright$  sample the current word
until  $y_m = \blacklozenge$             $\triangleright$  terminate when the stop symbol is generated

```

---

3592    Naïve Bayes was introduced as a generative model — a probabilistic story that ex-  
 3593    plains the observed data as well as the hidden label. A similar story can be constructed  
 3594    for probabilistic sequence labeling: first, the tags are drawn from a prior distribution; next,  
 3595    the tokens are drawn from a conditional likelihood. However, for inference to be tractable,  
 3596    additional independence assumptions are required. First, the probability of each token  
 3597    depends only on its tag, and not on any other element in the sequence:

$$p(w | y) = \prod_{m=1}^M p(w_m | y_m). \quad [7.32]$$

3598    Second, each tag  $y_m$  depends only on its predecessor,

$$p(y) = \prod_{m=1}^M p(y_m | y_{m-1}), \quad [7.33]$$

3599    where  $y_0 = \diamond$  in all cases. Due to this **Markov assumption**, probabilistic sequence labeling  
 3600    models are known as **hidden Markov models** (HMMs).

3601    The generative process for the hidden Markov model is shown in Algorithm 12. Given  
 3602    the parameters  $\lambda$  and  $\phi$ , we can compute  $p(w, y)$  for any token sequence  $w$  and tag se-  
 3603    quence  $y$ . The HMM is often represented as a **graphical model** (Wainwright and Jordan,  
 3604    2008), as shown in Figure 7.2. This representation makes the independence assumptions  
 3605    explicit: if a variable  $v_1$  is probabilistically conditioned on another variable  $v_2$ , then there  
 3606    is an arrow  $v_2 \rightarrow v_1$  in the diagram. If there are no arrows between  $v_1$  and  $v_2$ , they  
 3607    are **conditionally independent**, given each variable's **Markov blanket**. In the hidden  
 3608    Markov model, the Markov blanket for each tag  $y_m$  includes the “parent”  $y_{m-1}$ , and the  
 3609    “children”  $y_{m+1}$  and  $w_m$ .<sup>3</sup>

3610    It is important to reflect on the implications of the HMM independence assumptions.  
 3611    A non-adjacent pair of tags  $y_m$  and  $y_n$  are conditionally independent; if  $m < n$  and we  
 3612    are given  $y_{n-1}$ , then  $y_m$  offers no additional information about  $y_n$ . However, if we are  
 3613    not given any information about the tags in a sequence, then all tags are probabilistically  
 3614    coupled.

---

<sup>3</sup>In general graphical models, a variable's Markov blanket includes its parents, children, and its children's other parents (Murphy, 2012).

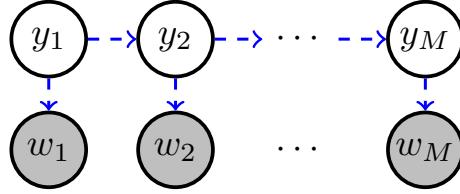


Figure 7.2: Graphical representation of the hidden Markov model. Arrows indicate probabilistic dependencies.

### 3615 7.4.1 Estimation

3616 The hidden Markov model has two groups of parameters:

3617 **Emission probabilities.** The probability  $p_e(w_m | y_m; \phi)$  is the emission probability, since  
3618 the words are treated as probabilistically “emitted”, conditioned on the tags.

3619 **Transition probabilities.** The probability  $p_t(y_m | y_{m-1}; \lambda)$  is the transition probability,  
3620 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,

$$\begin{aligned}\phi_{k,i} &\triangleq \Pr(W_m = i | Y_m = k) = \frac{\text{count}(W_m = i, Y_m = k)}{\text{count}(Y_m = k)} \\ \lambda_{k,k'} &\triangleq \Pr(Y_m = k' | Y_{m-1} = k) = \frac{\text{count}(Y_m = k', Y_{m-1} = k)}{\text{count}(Y_{m-1} = k)}.\end{aligned}$$

3621 Smoothing is more important for the emission probability than the transition probability,  
3622 because the vocabulary is much larger than the number of tags.

### 3623 7.4.2 Inference

3624 The goal of inference in the hidden Markov model is to find the highest probability tag  
3625 sequence,

$$\hat{y} = \underset{y}{\operatorname{argmax}} p(y | w). \quad [7.34]$$

3626 As in Naïve Bayes, it is equivalent to find the tag sequence with the highest *log*-probability,  
3627 since the logarithm is a monotonically increasing function. It is furthermore equivalent  
3628 to maximize the joint probability  $p(y, w) = p(y | w) \times p(w) \propto p(y | w)$ , which is pro-  
3629 portional to the conditional probability. Putting these observations together, the inference

problem can be reformulated as,

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}} \log p(\mathbf{y}, \mathbf{w}). \quad [7.35]$$

We can now apply the HMM independence assumptions:

$$\log p(\mathbf{y}, \mathbf{w}) = \log p(\mathbf{y}) + \log p(\mathbf{w} \mid \mathbf{y}) \quad [7.36]$$

$$= \sum_{m=1}^{M+1} \log p_Y(y_m \mid y_{m-1}) + \log p_{W|Y}(w_m \mid y_m) \quad [7.37]$$

$$= \sum_{m=1}^{M+1} \log \lambda_{y_m, y_{m-1}} + \log \phi_{y_m, w_m} \quad [7.38]$$

$$= \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}), \quad [7.39]$$

where,

$$s_m(y_m, y_{m-1}) \triangleq \log \lambda_{y_m, y_{m-1}} + \log \phi_{y_m, w_m}, \quad [7.40]$$

and,

$$\phi_{\diamond, w} = \begin{cases} 1, & w = \blacksquare \\ 0, & \text{otherwise,} \end{cases} \quad [7.41]$$

which ensures that the stop tag  $\diamond$  can only be applied to the final token  $\blacksquare$ .

This derivation shows that HMM inference can be viewed as an application of the Viterbi decoding algorithm, given an appropriately defined scoring function. The local score  $s_m(y_m, y_{m-1})$  can be interpreted probabilistically,

$$s_m(y_m, y_{m-1}) = \log p_y(y_m \mid y_{m-1}) + \log p_{w|y}(w_m \mid y_m) \quad [7.42]$$

$$= \log p(y_m, w_m \mid y_{m-1}). \quad [7.43]$$

Now recall the definition of the Viterbi variables,

$$v_m(y_m) = \max_{y_{m-1}} s_m(y_m, y_{m-1}) + v_{m-1}(y_{m-1}) \quad [7.44]$$

$$= \max_{y_{m-1}} \log p(y_m, w_m \mid y_{m-1}) + v_{m-1}(y_{m-1}). \quad [7.45]$$

By setting  $v_{m-1}(y_{m-1}) = \max_{\mathbf{y}_{1:m-2}} \log p(\mathbf{y}_{1:m-2}, \mathbf{w}_{1:m-1})$ , we obtain the recurrence,

$$v_m(y_m) = \max_{y_{m-1}} \log p(y_m, w_m \mid y_{m-1}) + \max_{\mathbf{y}_{1:m-2}} \log p(\mathbf{y}_{1:m-2}, \mathbf{w}_{1:m-1}) \quad [7.46]$$

$$= \max_{\mathbf{y}_{1:m-1}} \log p(y_m, w_m \mid y_{m-1}) + \log p(\mathbf{y}_{1:m-1}, \mathbf{w}_{1:m-1}) \quad [7.47]$$

$$= \max_{\mathbf{y}_{1:m-1}} \log p(\mathbf{y}_{1:m-1}, \mathbf{w}_{1:m-1}). \quad [7.48]$$

In words, the Viterbi variable  $v_m(y_m)$  is the log probability of the best tag sequence ending in  $y_m$ , joint with the word sequence  $w_{1:m}$ . The log probability of the best complete tag sequence is therefore,

$$\max_{\mathbf{y}_{1:M}} \log p(\mathbf{y}_{1:M+1}, \mathbf{w}_{1:M+1}) = v_{M+1}(\spadesuit) \quad [7.49]$$

**\*Viterbi as an example of the max-product algorithm** The Viterbi algorithm can also be implemented using probabilities, rather than log-probabilities. In this case, each  $v_m(y_m)$  is equal to,

$$v_m(y_m) = \max_{\mathbf{y}_{1:m-1}} p(\mathbf{y}_{1:m-1}, y_m, \mathbf{w}_{1:m}) \quad [7.50]$$

$$= \max_{y_{m-1}} p(y_m, w_m | y_{m-1}) \times \max_{\mathbf{y}_{1:m-2}} p(\mathbf{y}_{1:m-2}, y_{m-1}, \mathbf{w}_{1:m-1}) \quad [7.51]$$

$$= \max_{y_{m-1}} p(y_m, w_m | y_{m-1}) \times v_{m-1}(y_{m-1}) \quad [7.52]$$

$$= p_{w|y}(w_m | y_m) \times \max_{y_{m-1}} p_y(y_m | y_{m-1}) \times v_{m-1}(y_{m-1}). \quad [7.53]$$

3633 Each Viterbi variable is computed by *maximizing* over a set of *products*. Thus, the Viterbi  
 3634 algorithm is a special case of the **max-product algorithm** for inference in graphical mod-  
 3635 els (Wainwright and Jordan, 2008). However, the product of probabilities tends towards  
 3636 zero over long sequences, so the log-probability version of Viterbi is recommended in  
 3637 practical implementations.

## 3638 7.5 Discriminative sequence labeling with features

3639 Today, hidden Markov models are rarely used for supervised sequence labeling. This is  
 3640 because HMMs are limited to only two phenomena:

- 3641 • word-tag compatibility, via the emission probability  $p_{W|Y}(w_m | y_m)$ ;
- 3642 • local context, via the transition probability  $p_Y(y_m | y_{m-1})$ .

3643 The Viterbi algorithm permits the inclusion of richer information in the local scoring func-  
 3644 tion  $\psi(\mathbf{w}_{1:M}, y_m, y_{m-1}, m)$ , which can be defined as a weighted sum of arbitrary local *fea-*  
 3645 *tures*,

$$\psi(\mathbf{w}, y_m, y_{m-1}, m) = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m), \quad [7.54]$$

3646 where  $\mathbf{f}$  is a locally-defined feature function, and  $\boldsymbol{\theta}$  is a vector of weights.

The local decomposition of the scoring function  $\Psi$  is reflected in a corresponding decomposition of the feature function:

$$\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.55]$$

$$= \theta \cdot \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.56]$$

$$= \theta \cdot \sum_{m=1}^{M+1} \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.57]$$

$$= \theta \cdot \mathbf{f}^{(\text{global})}(\mathbf{w}, \mathbf{y}_{1:M}), \quad [7.58]$$

3647 where  $\mathbf{f}^{(\text{global})}(\mathbf{w}, \mathbf{y})$  is a global feature vector, which is a sum of local feature vectors,

$$\mathbf{f}^{(\text{global})}(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \mathbf{f}(\mathbf{w}_{1:M}, y_m, y_{m-1}, m), \quad [7.59]$$

3648 with  $y_{M+1} = \diamond$  and  $y_0 = \diamond$  by construction.

3649 Let's now consider what additional information these features might encode.

3650 **Word affix features.** Consider the problem of part-of-speech tagging on the first four  
3651 lines of the poem *Jabberwocky* (Carroll, 1917):

3652 (7.3) 'Twas brillig, and the slithy toves  
3653 Did gyre and gimble in the wabe:  
3654 All mimsy were the borogoves,  
3655 And the mome raths outgrabe.

3656 Many of these words were made up by the author of the poem, so a corpus would offer  
3657 no information about their probabilities of being associated with any particular part of  
3658 speech. Yet it is not so hard to see what their grammatical roles might be in this passage.  
3659 Context helps: for example, the word *slithy* follows the determiner *the*, so it is probably a  
3660 noun or adjective. Which do you think is more likely? The suffix *-thy* is found in a number  
3661 of adjectives, like *frothy*, *healthy*, *pithy*, *worthy*. It is also found in a handful of nouns — e.g.,  
3662 *apathy*, *sympathy* — but nearly all of these have the longer coda *-pathy*, unlike *slithy*. So the  
3663 suffix gives some evidence that *slithy* is an adjective, and indeed it is: later in the text we  
3664 find that it is a combination of the adjectives *lithe* and *slimy*.<sup>4</sup>

---

<sup>4</sup>Morphology is the study of how words are formed from smaller linguistic units. Computational approaches to morphological analysis are touched on in chapter 9; Bender (2013) provides a good overview of the underlying linguistic principles.

3665 **Fine-grained context.** The hidden Markov model captures contextual information in the  
 3666 form of part-of-speech tag bigrams. But sometimes, the necessary contextual information  
 3667 is more specific. Consider the noun phrases *this fish* and *these fish*. Many part-of-speech  
 3668 tagsets distinguish between singular and plural nouns, but do not distinguish between  
 3669 singular and plural determiners.<sup>5</sup> A hidden Markov model would be unable to correctly  
 3670 label *fish* as singular or plural in both of these cases, because it only has access to two  
 3671 features: the preceding tag (determiner in both cases) and the word (*fish* in both cases).  
 3672 The classification-based tagger discussed in § 7.1 had the ability to use preceding and suc-  
 3673 ceeding words as features, and it can also be incorporated into a Viterbi-based sequence  
 3674 labeler as a local feature.

**Example** Consider the tagging D J N (determiner, adjective, noun) for the sequence *the slithy toves*, so that

$$\mathbf{w} = \text{the slithy toves}$$

$$\mathbf{y} = \text{D J N}.$$

Let's create the feature vector for this example, assuming that we have word-tag features (indicated by  $W$ ), tag-tag features (indicated by  $T$ ), and suffix features (indicated by  $M$ ). You can assume that you have access to a method for extracting the suffix *-thy* from *slithy*, *-es* from *toves*, and  $\emptyset$  from *the*, indicating that this word has no suffix.<sup>6</sup> The resulting feature vector is,

$$\begin{aligned} f(\text{the slithy toves, D J N}) &= f(\text{the slithy toves, D}, \diamond, 1) \\ &\quad + f(\text{the slithy toves, J}, D, 2) \\ &\quad + f(\text{the slithy toves, N}, J, 3) \\ &\quad + f(\text{the slithy toves}, \blacklozenge, N, 4) \\ &= \{(T : \diamond, D), (W : \text{the}, D), (M : \emptyset, D), \\ &\quad (T : D, J), (W : \text{slithy}, J), (M : \text{-thy}, J), \\ &\quad (T : J, N), (W : \text{toves}, N), (M : \text{-es}, N) \\ &\quad (T : N, \blacklozenge)\}. \end{aligned}$$

3675 These examples show that local features can incorporate information that lies beyond  
 3676 the scope of a hidden Markov model. Because the features are local, it is possible to apply  
 3677 the Viterbi algorithm to identify the optimal sequence of tags. The remaining question

<sup>5</sup>For example, the Penn Treebank tagset follows these conventions.

<sup>6</sup>Such 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*.

3678 is how to estimate the weights on these features. § 2.2 presented three main types of  
 3679 discriminative classifiers: perceptron, support vector machine, and logistic regression.  
 3680 Each of these classifiers has a structured equivalent, enabling it to be trained from labeled  
 3681 sequences rather than individual tokens.

3682 **7.5.1 Structured perceptron**

The perceptron classifier is trained by increasing the weights for features that are associated with the correct label, and decreasing the weights for features that are associated with incorrectly predicted labels:

$$\hat{y} = \operatorname{argmax}_{y \in \mathcal{Y}} \theta \cdot f(\mathbf{x}, y) \quad [7.60]$$

$$\theta^{(t+1)} \leftarrow \theta^{(t)} + f(\mathbf{x}, y) - f(\mathbf{x}, \hat{y}). \quad [7.61]$$

We can apply exactly the same update in the case of structure prediction,

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} \theta \cdot f(\mathbf{w}, \mathbf{y}) \quad [7.62]$$

$$\theta^{(t+1)} \leftarrow \theta^{(t)} + f(\mathbf{w}, \mathbf{y}) - f(\mathbf{w}, \hat{\mathbf{y}}). \quad [7.63]$$

3683 This learning algorithm is called **structured perceptron**, because it learns to predict the  
 3684 structured output  $\mathbf{y}$ . The only difference is that instead of computing  $\hat{y}$  by enumerating  
 3685 the entire set  $\mathcal{Y}$ , the Viterbi algorithm is used to efficiently search the set of possible tag-  
 3686 gings,  $\mathcal{Y}^M$ . Structured perceptron can be applied to other structured outputs as long as  
 3687 efficient inference is possible. As in perceptron classification, weight averaging is crucial  
 3688 to get good performance (see § 2.2.2).

**Example** For the example *they can fish*, suppose that the reference tag sequence is  $\mathbf{y}^{(i)} =$   
 N V V, but the tagger incorrectly returns the tag sequence  $\hat{\mathbf{y}} = \text{N V N}$ . Assuming a model  
 with features for emissions  $(w_m, y_m)$  and transitions  $(y_{m-1}, y_m)$ , the corresponding structured  
 perceptron update is:

$$\theta_{(fish,V)} \leftarrow \theta_{(fish,V)} + 1, \quad \theta_{(fish,N)} \leftarrow \theta_{(fish,N)} - 1 \quad [7.64]$$

$$\theta_{(V,V)} \leftarrow \theta_{(V,V)} + 1, \quad \theta_{(V,N)} \leftarrow \theta_{(V,N)} - 1 \quad [7.65]$$

$$\theta_{(V,\blacklozenge)} \leftarrow \theta_{(V,\blacklozenge)} + 1, \quad \theta_{(N,\blacklozenge)} \leftarrow \theta_{(N,\blacklozenge)} - 1. \quad [7.66]$$

3689 **7.5.2 Structured support vector machines**

3690 Large-margin classifiers such as the support vector machine improve on the perceptron by  
 3691 pushing the classification boundary away from the training instances. The same idea can

3692 be applied to sequence labeling. A support vector machine in which the output is a struc-  
 3693 tured object, such as a sequence, is called a **structured support vector machine** (Tsochan-  
 3694 taridis et al., 2004).<sup>7</sup>

3695 In classification, we formalized the large-margin constraint as,

$$\forall \mathbf{y} \neq \mathbf{y}^{(i)}, \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}^{(i)}) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}) \geq 1, \quad [7.67]$$

3696 requiring a margin of at least 1 between the scores for all labels  $\mathbf{y}$  that are not equal to the  
 3697 correct label  $\mathbf{y}^{(i)}$ . The weights  $\boldsymbol{\theta}$  are then learned by constrained optimization (see § 2.3.2).

3698 This idea can be applied to sequence labeling by formulating an equivalent set of con-  
 3699 straints for all possible labelings  $\mathcal{Y}(\mathbf{w})$  for an input  $\mathbf{w}$ . However, there are two problems.  
 3700 First, in sequence labeling, some predictions are more wrong than others: we may miss  
 3701 only one tag out of fifty, or we may get all fifty wrong. We would like our learning algo-  
 3702 rithm to be sensitive to this difference. Second, the number of constraints is equal to the  
 3703 number of possible labelings, which is exponentially large in the length of the sequence.

3704 The first problem can be addressed by adjusting the constraint to require larger mar-  
 3705 gins for more serious errors. Let  $c(\mathbf{y}^{(i)}, \hat{\mathbf{y}}) \geq 0$  represent the *cost* of predicting label  $\hat{\mathbf{y}}$  when  
 3706 the true label is  $\mathbf{y}^{(i)}$ . We can then generalize the margin constraint,

$$\forall \mathbf{y}, \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) \geq c(\mathbf{y}^{(i)}, \mathbf{y}). \quad [7.68]$$

3707 This cost-augmented margin constraint specializes to the constraint in Equation 7.67 if we  
 3708 choose the delta function  $c(\mathbf{y}^{(i)}, \mathbf{y}) = \delta((\mathbf{y}^{(i)} \neq \mathbf{y}))$ . A more expressive cost function is  
 3709 the **Hamming cost**,

$$c(\mathbf{y}^{(i)}, \mathbf{y}) = \sum_{m=1}^M \delta(y_m^{(i)} \neq y_m), \quad [7.69]$$

3710 which computes the number of errors in  $\mathbf{y}$ . By incorporating the cost function as the  
 3711 margin constraint, we require that the true labeling be separated from the alternatives by  
 3712 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:

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \neq \mathbf{y}^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) + c(\mathbf{y}^{(i)}, \mathbf{y}) \quad [7.70]$$

$$= \operatorname{argmax}_{\mathbf{y} \neq \mathbf{y}^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) + c(\mathbf{y}^{(i)}, \mathbf{y}), \quad [7.71]$$

---

<sup>7</sup>This 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.

3713 where in the second line we drop the term  $\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})$ , which is constant in  $\mathbf{y}$ .

We can now reformulate the margin constraint for sequence labeling,

$$\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) - \max_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} (\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}) + c(\mathbf{y}^{(i)}, \mathbf{y})) \geq 0. \quad [7.72]$$

3714 If the score for  $\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})$  is greater than the cost-augmented score for all alternatives,  
 3715 then the constraint will be met. The name “cost-augmented decoding” is due to the fact  
 3716 that the objective includes the standard decoding problem,  $\max_{\hat{\mathbf{y}} \in \mathcal{Y}(\mathbf{w})} \theta \cdot f(\mathbf{w}, \hat{\mathbf{y}})$ , plus  
 3717 an additional term for the cost. Essentially, we want to train against predictions that are  
 3718 strong and wrong: they should score highly according to the model, yet incur a large loss  
 3719 with respect to the ground truth. Training adjusts the weights to reduce the score of these  
 3720 predictions.

3721 For cost-augmented decoding to be tractable, the cost function must decompose into  
 3722 local parts, just as the feature function  $f(\cdot)$  does. The Hamming cost, defined above,  
 3723 obeys this property. To perform cost-augmented decoding using the Hamming cost, we  
 3724 need only to add features  $f_m(y_m) = \delta(y_m \neq y_m^{(i)})$ , and assign a constant weight of 1 to  
 3725 these features. Decoding can then be performed using the Viterbi algorithm.<sup>8</sup>

As with large-margin classifiers, it is possible to formulate the learning problem in an unconstrained form, by combining a regularization term on the weights and a Lagrangian for the constraints:

$$\min_{\theta} \frac{1}{2} \|\theta\|_2^2 - C \left( \sum_i \theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) - \max_{\mathbf{y} \in \mathcal{Y}(\mathbf{w}^{(i)})} [\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}) + c(\mathbf{y}^{(i)}, \mathbf{y})] \right), \quad [7.73]$$

3726 In this formulation,  $C$  is a parameter that controls the tradeoff between the regularization  
 3727 term and the margin constraints. A number of optimization algorithms have been  
 3728 proposed for structured support vector machines, some of which are discussed in § 2.3.2.  
 3729 An empirical comparison by Kummerfeld et al. (2015) shows that stochastic subgradient  
 3730 descent — which is essentially a cost-augmented version of the structured perceptron —  
 3731 is highly competitive.

### 3732 7.5.3 Conditional random fields

3733 The **conditional random field** (CRF; Lafferty et al., 2001) is a conditional probabilistic  
 3734 model for sequence labeling; just as structured perceptron is built on the perceptron clas-  
 3735 sifier, conditional random fields are built on the logistic regression classifier.<sup>9</sup> The basic

---

<sup>8</sup>Are 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.

<sup>9</sup>The 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

3736 probability model is,

$$p(\mathbf{y} \mid \mathbf{w}) = \frac{\exp(\Psi(\mathbf{w}, \mathbf{y}))}{\sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp(\Psi(\mathbf{w}, \mathbf{y}'))}. \quad [7.74]$$

3737 This is almost identical to logistic regression, but because the label space is now tag  
 3738 sequences, we require efficient algorithms for both **decoding** (searching for the best tag  
 3739 sequence given a sequence of words  $\mathbf{w}$  and a model  $\theta$ ) and for **normalizing** (summing  
 3740 over all tag sequences). These algorithms will be based on the usual locality assumption  
 3741 on the scoring function,  $\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m)$ .

3742 **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  $p(\mathbf{y} \mid \mathbf{w})$ ,

$$\begin{aligned} \hat{\mathbf{y}} &= \operatorname{argmax}_{\mathbf{y}} \log p(\mathbf{y} \mid \mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{y}} \Psi(\mathbf{y}, \mathbf{w}) - \log \sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp \Psi(\mathbf{y}', \mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{y}} \Psi(\mathbf{y}, \mathbf{w}) = \operatorname{argmax}_{\mathbf{y}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}). \end{aligned}$$

3743 This is identical to the decoding problem for structured perceptron, so the same Viterbi  
 3744 recurrence as defined in Equation 7.22 can be used.

3745 **7.5.3.2 Learning in CRFs**

As with logistic regression, the weights  $\theta$  are learned by minimizing the regularized negative log-probability,

$$\ell = \frac{\lambda}{2} \|\theta\|^2 - \sum_{i=1}^N \log p(\mathbf{y}^{(i)} \mid \mathbf{w}^{(i)}; \theta) \quad [7.75]$$

$$= \frac{\lambda}{2} \|\theta\|^2 - \sum_{i=1}^N \theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) + \log \sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w}^{(i)})} \exp (\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}')), \quad [7.76]$$

---

more generally, cliques) of variables in a **factor graph**. In sequence labeling, the pairs of variables include all adjacent tags ( $y_m, y_{m-1}$ ). The probability is *conditioned* on the words  $\mathbf{w}$ , which are always observed, motivating the term “conditional” in the name.

3746 where  $\lambda$  controls the amount of regularization. The final term in Equation 7.76 is a sum  
 3747 over all possible labelings. This term is the log of the denominator in Equation 7.74, some-  
 3748 times known as the **partition function**.<sup>10</sup> There are  $|\mathcal{Y}|^M$  possible labelings of an input of  
 3749 size  $M$ , so we must again exploit the decomposition of the scoring function to compute  
 3750 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,

$$\sum_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} \Psi(\mathbf{w}, \mathbf{y}) = \sum_{\mathbf{y}_{1:M}} s_{M+1}(\blacklozenge, y_M) \prod_{m=1}^M s_m(y_m, y_{m-1}) \quad [7.82]$$

$$= \alpha_{M+1}(\blacklozenge). \quad [7.83]$$

---

<sup>10</sup>The terminology of “potentials” and “partition functions” comes from statistical mechanics (Bishop, 2006).

The conditional log-likelihood can be rewritten,

$$\ell = \frac{\lambda}{2} \|\boldsymbol{\theta}\|^2 - \sum_{i=1}^N \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) + \log \alpha_{M+1}(\blacklozenge). \quad [7.84]$$

- 3751 Probabilistic programming environments, such as `Torch` (Collobert et al., 2011) and `dynet` (Neu-  
 3752 big et al., 2017), can compute the gradient of this objective using automatic differentiation.  
 3753 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:

$$\frac{d\ell}{d\theta_j} = \lambda \theta_j + \sum_{i=1}^N E[f_j(\mathbf{w}^{(i)}, \mathbf{y})] - f_j(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}), \quad [7.85]$$

- 3754 where  $f_j(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})$  refers to the count of feature  $j$  for token sequence  $\mathbf{w}^{(i)}$  and tag se-  
 3755 quence  $\mathbf{y}^{(i)}$ . The expected feature counts are computed “under the hood” when automatic  
 3756 differentiation is applied to Equation 7.84 (Eisner, 2016).

- 3757 Before the widespread use of automatic differentiation, it was common to compute  
 3758 the feature expectations from marginal tag probabilities  $p(y_m | \mathbf{w})$ . These marginal prob-  
 3759 abilities are sometimes useful on their own, and can be computed using the **forward-**  
 3760 **backward algorithm**. This algorithm combines the forward recurrence with an equivalent  
 3761 **backward recurrence**, which traverses the input from  $w_M$  back to  $w_1$ .

### 3762 7.5.3.3 \*Forward-backward algorithm

Marginal probabilities over tag bigrams can be written as,<sup>11</sup>

$$\Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}) = \frac{\sum_{\mathbf{y}: Y_m=k, Y_{m-1}=k'} \prod_{n=1}^M \exp s_n(y_n, y_{n-1})}{\sum_{\mathbf{y}'} \prod_{n=1}^M \exp s_n(y'_n, y'_{n-1})}. \quad [7.86]$$

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

---

<sup>11</sup>Recall the notational convention of upper-case letters for random variables, e.g.  $Y_m$ , and lower case letters for specific values, e.g.,  $y_m$ , so that  $Y_m = k$  is interpreted as the event of random variable  $Y_m$  taking the value  $k$ .

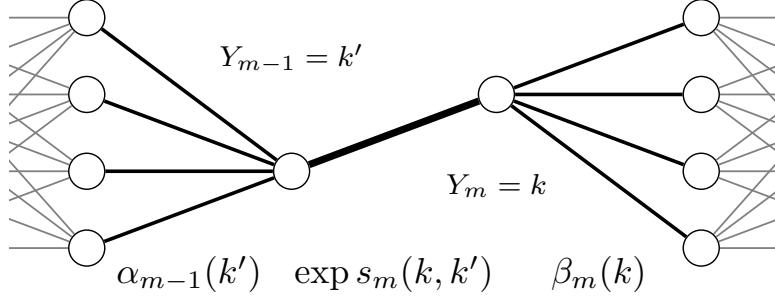


Figure 7.3: A schematic illustration of the computation of the marginal probability  $\Pr(Y_{m-1} = k', Y_m = k)$ , using the forward score  $\alpha_{m-1}(k')$  and the backward score  $\beta_m(k)$ .

transition  $(Y_{m-1} = k') \rightarrow (Y_m = k)$ ; and the suffixes  $\mathbf{y}_{m:M}$ , beginning with the tag  $Y_m = k$ :

$$\sum_{\mathbf{y}: Y_m = k, Y_{m-1} = k'} \prod_{n=1}^M \exp s_n(y_n, y_{n-1}) = \sum_{\mathbf{y}_{1:m-1}: Y_{m-1} = k'} \prod_{n=1}^{m-1} \exp s_n(y_n, y_{n-1}) \times \exp s_m(k, k') \times \sum_{\mathbf{y}_{m:M}: Y_m = k} \prod_{n=m+1}^{M+1} \exp s_n(y_n, y_{n-1}). \quad [7.87]$$

The result is product of three terms: a score that sums over all the ways to get to the position  $(Y_{m-1} = k')$ , a score for the transition from  $k'$  to  $k$ , and a score that sums over all the ways of finishing the sequence from  $(Y_m = k)$ . The first term of Equation 7.87 is equal to the **forward variable**,  $\alpha_{m-1}(k')$ . The third term — the sum over ways to finish the sequence — can also be defined recursively, this time moving over the trellis from right to left, which is known as the **backward recurrence**:

$$\beta_m(k) \triangleq \sum_{\mathbf{y}_{m:M}: Y_m = k} \prod_{n=m}^{M+1} \exp s_n(y_n, y_{n-1}) \quad [7.88]$$

$$= \sum_{k' \in \mathcal{Y}} \exp s_{m+1}(k', k) \sum_{\mathbf{y}_{m+1:M}: Y_m = k'} \prod_{n=m+1}^{M+1} \exp s_n(y_n, y_{n-1}) \quad [7.89]$$

$$= \sum_{k' \in \mathcal{Y}} \exp s_{m+1}(k', k) \times \beta_{m+1}(k'). \quad [7.90]$$

<sup>3763</sup> To understand this computation, compare with the forward recurrence in Equation 7.81.

In practice, numerical stability demands that we work in the log domain,

$$\log \alpha_m(k) = \log \sum_{k' \in \mathcal{Y}} \exp (\log s_m(k, k') + \log \alpha_{m-1}(k')) \quad [7.91]$$

$$\log \beta_{m-1}(k) = \log \sum_{k' \in \mathcal{Y}} \exp (\log s_m(k', k) + \log \beta_m(k')). \quad [7.92]$$

The application of the forward and backward probabilities is shown in Figure 7.3. Both the forward and backward recurrences operate on the trellis, which implies a space complexity  $\mathcal{O}(MK)$ . Because both recurrences require computing a sum over  $K$  terms at each node in the trellis, their time complexity is  $\mathcal{O}(MK^2)$ .

## 7.6 Neural sequence labeling

In neural network approaches to sequence labeling, we construct a vector representation for each tagging decision, based on the word and its context. Neural networks can perform tagging as a per-token classification decision, or they can be combined with the Viterbi algorithm to tag the entire sequence globally.

### 7.6.1 Recurrent neural networks

Recurrent neural networks (RNNs) were introduced in chapter 6 as a language modeling technique, in which the context at token  $m$  is summarized by a recurrently-updated vector,

$$\mathbf{h}_m = g(\mathbf{x}_m, \mathbf{h}_{m-1}), \quad m = 1, 2, \dots, M,$$

where  $\mathbf{x}_m$  is the vector **embedding** of the token  $w_m$  and the function  $g$  defines the recurrence. The starting condition  $\mathbf{h}_0$  is an additional parameter of the model. The long short-term memory (LSTM) is a more complex recurrence, in which a memory cell is through a series of gates, avoiding repeated application of the non-linearity. Despite these bells and whistles, both models share the basic architecture of recurrent updates across a sequence, and both will be referred to as RNNs here.

A straightforward application of RNNs to sequence labeling is to score each tag  $y_m$  as a linear function of  $\mathbf{h}_m$ :

$$\psi_m(y) = \beta_y \cdot \mathbf{h}_m \quad [7.93]$$

$$\hat{y}_m = \underset{y}{\operatorname{argmax}} \psi_m(y). \quad [7.94]$$

The score  $\psi_m(y)$  can also be converted into a probability distribution using the usual softmax operation,

$$p(y | \mathbf{w}_{1:m}) = \frac{\exp \psi_m(y)}{\sum_{y' \in \mathcal{Y}} \exp \psi_m(y')}. \quad [7.95]$$

3782 Using this transformation, it is possible to train the tagger from the negative log-likelihood  
 3783 of the tags, as in a conditional random field. Alternatively, a hinge loss or margin loss  
 3784 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:

$$\overleftarrow{\mathbf{h}}_m = g(\mathbf{x}_m, \overleftarrow{\mathbf{h}}_{m+1}), \quad m = 1, 2, \dots, M. \quad [7.96]$$

3785 The hidden states of the left-to-right RNN are denoted  $\overrightarrow{\mathbf{h}}_m$ . The left-to-right and right-to-  
 3786 left vectors are concatenated,  $\mathbf{h}_m = [\overleftarrow{\mathbf{h}}_m; \overrightarrow{\mathbf{h}}_m]$ . The scoring function in Equation 7.93 is  
 3787 applied to this concatenated vector.

3788 Bidirectional RNN tagging has several attractive properties. Ideally, the representa-  
 3789 tion  $\mathbf{h}_m$  summarizes the useful information from the surrounding context, so that it is not  
 3790 necessary to design explicit features to capture this information. If the vector  $\mathbf{h}_m$  is an ad-  
 3791 equate summary of this context, then it may not even be necessary to perform the tagging  
 3792 jointly: in general, the gains offered by joint tagging of the entire sequence are diminished  
 3793 as the individual tagging model becomes more powerful. Using backpropagation, the  
 3794 word vectors  $\mathbf{x}$  can be trained “end-to-end”, so that they capture word properties that are  
 3795 useful for the tagging task. Alternatively, if limited labeled data is available, we can use  
 3796 word embeddings that are “pre-trained” from unlabeled data, using a language modeling  
 3797 objective (as in § 6.3) or a related word embedding technique (see chapter 14). It is even  
 3798 possible to combine both fine-tuned and pre-trained embeddings in a single model.

3799 **Neural structure prediction** The bidirectional recurrent neural network incorporates in-  
 3800 formation from throughout the input, but each tagging decision is made independently.  
 3801 In some sequence labeling applications, there are very strong dependencies between tags:  
 3802 it may even be impossible for one tag to follow another. In such scenarios, the tagging  
 3803 decision must be made jointly across the entire sequence.

3804 Neural sequence labeling can be combined with the Viterbi algorithm by defining the  
 3805 local scores as:

$$s_m(y_m, y_{m-1}) = \beta_{y_m} \cdot \mathbf{h}_m + \eta_{y_{m-1}, y_m}, \quad [7.97]$$

3806 where  $\mathbf{h}_m$  is the RNN hidden state,  $\beta_{y_m}$  is a vector associated with tag  $y_m$ , and  $\eta_{y_{m-1}, y_m}$   
 3807 is a scalar parameter for the tag transition  $(y_{m-1}, y_m)$ . These local scores can then be  
 3808 incorporated into the Viterbi algorithm for inference, and into the forward algorithm for  
 3809 training. This model is shown in Figure 7.4. It can be trained from the conditional log-  
 3810 likelihood objective defined in Equation 7.76, backpropagating to the tagging parameters

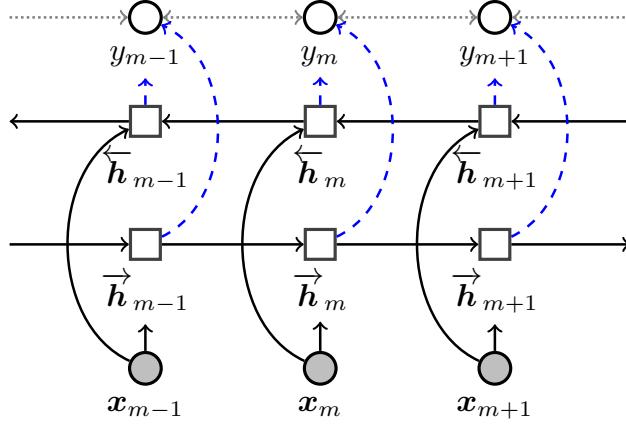


Figure 7.4: Bidirectional LSTM for sequence labeling. The solid lines indicate computation, the dashed lines indicate probabilistic dependency, and the dotted lines indicate the optional additional probabilistic dependencies between labels in the biLSTM-CRF.

3811  $\beta$  and  $\eta$ , as well as the parameters of the RNN. This model is called the **LSTM-CRF**, due  
 3812 to its combination of aspects of the long short-term memory and conditional random field  
 3813 models (Huang et al., 2015).

3814 The LSTM-CRF is especially effective on the task of **named entity recognition** (Lample  
 3815 et al., 2016), a sequence labeling task that is described in detail in § 8.3. This task has strong  
 3816 dependencies between adjacent tags, so structure prediction is especially important.

### 3817 7.6.2 Character-level models

3818 As in language modeling, rare and unseen words are a challenge: if we encounter a word  
 3819 that was not in the training data, then there is no obvious choice for the word embed-  
 3820 ding  $x_m$ . One solution is to use a generic **unseen word** embedding for all such words.  
 3821 However, in many cases, properties of unseen words can be guessed from their spellings.  
 3822 For example, *whimsical* does not appear in the Universal Dependencies (UD) English Tree-  
 3823 bank, yet the suffix *-al* makes it likely to be adjective; by the same logic, *unflinchingly* is  
 3824 likely to be an adverb, and *barnacle* is likely to be a noun.

3825 In feature-based models, these morphological properties were handled by suffix fea-  
 3826 tures; in a neural network, they can be incorporated by constructing the embeddings of  
 3827 unseen words from their spellings or morphology. One way to do this is to incorporate  
 3828 an additional layer of bidirectional RNNs, one for each word in the vocabulary (Ling  
 3829 et al., 2015). For each such character-RNN, the inputs are the characters, and the output  
 3830 is the concatenation of the final states of the left-facing and right-facing passes,  $\phi_w =$

[ $\vec{h}_{N_w}^{(w)}; \overleftarrow{h}_0^{(w)}$ ], where  $\vec{h}_{N_w}^{(w)}$  is the final state of the right-facing pass for word  $w$ , and  $N_w$  is the number of characters in the word. The character RNN model is trained by back-propagation from the tagging objective. On the test data, the trained RNN is applied to out-of-vocabulary words (or all words), yielding inputs to the word-level tagging RNN. Other approaches to compositional word embeddings are described in § 14.7.1.

### 7.6.3 Convolutional Neural Networks for Sequence Labeling

One disadvantage of recurrent neural networks is that the architecture requires iterating through the sequence of inputs and predictions: each hidden vector  $h_m$  must be computed from the previous hidden vector  $h_{m-1}$ , before predicting the tag  $y_m$ . These iterative computations are difficult to parallelize, and fail to exploit the speedups offered by **graphics processing units (GPUs)** on operations such as matrix multiplication. **Convolutional neural networks** achieve better computational performance by predicting each label  $y_m$  from a set of matrix operations on the neighboring word embeddings,  $x_{m-k:m+k}$  (Collobert et al., 2011). Because there is no hidden state to update, the predictions for each  $y_m$  can be computed in parallel. For more on convolutional neural networks, see § 3.4. Character-based word embeddings can also be computed using convolutional neural networks (Santos and Zadrozny, 2014).

## 7.7 \*Unsupervised sequence labeling

In unsupervised sequence labeling, the goal is to induce a hidden Markov model from a corpus of *unannotated* text ( $w^{(1)}, w^{(2)}, \dots, w^{(N)}$ ), where each  $w^{(i)}$  is a sequence of length  $M^{(i)}$ . This is an example of the general problem of **structure induction**, which is the unsupervised version of structure prediction. The tags that result from unsupervised sequence labeling might be useful for some downstream task, or they might help us to better understand the language’s inherent structure. For part-of-speech tagging, it is common to use a tag dictionary that lists the allowed tags for each word, simplifying the problem (Christodoulopoulos et al., 2010).

Unsupervised learning in hidden Markov models can be performed using the **Baum-Welch algorithm**, which combines the forward-backward algorithm (§ 7.5.3.3) with expectation-maximization (EM; § 5.1.2). In the M-step, the HMM parameters from expected counts:

$$\Pr(W = i \mid Y = k) = \phi_{k,i} = \frac{E[\text{count}(W = i, Y = k)]}{E[\text{count}(Y = k)]}$$

$$\Pr(Y_m = k \mid Y_{m-1} = k') = \lambda_{k',k} = \frac{E[\text{count}(Y_m = k, Y_{m-1} = k')]}{E[\text{count}(Y_{m-1} = k')]} \quad 3856$$

3857 The expected counts are computed in the E-step, using the forward and backward  
 3858 recurrences. The local scores follow the usual definition for hidden Markov models,

$$s_m(k, k') = \log p_E(w_m | Y_m = k; \phi) + \log p_T(Y_m = k | Y_{m-1} = k'; \lambda). \quad [7.98]$$

The expected transition counts for a single instance are,

$$E[\text{count}(Y_m = k, Y_{m-1} = k') | \mathbf{w}] = \sum_{m=1}^M \Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}) \quad [7.99]$$

$$= \frac{\sum_{\mathbf{y}: Y_m=k, Y_{m-1}=k'} \prod_{n=1}^M \exp s_n(y_n, y_{n-1})}{\sum_{\mathbf{y}'} \prod_{n=1}^M \exp s_n(y'_n, y'_{n-1})}. \quad [7.100]$$

As described in § 7.5.3.3, these marginal probabilities can be computed from the forward-backward recurrence,

$$\Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}) = \frac{\alpha_{m-1}(k') \times s_m(k, k') \times \beta_m(k)}{\alpha_{M+1}(\blacklozenge)}. \quad [7.101]$$

In a hidden Markov model, each element of the forward-backward computation has a special interpretation:

$$\alpha_{m-1}(k') = p(Y_{m-1} = k', \mathbf{w}_{1:m-1}) \quad [7.102]$$

$$s_m(k, k') = p(Y_m = k, w_m | Y_{m-1} = k') \quad [7.103]$$

$$\beta_m(k) = p(\mathbf{w}_{m+1:M} | Y_m = k). \quad [7.104]$$

Applying the conditional independence assumptions of the hidden Markov model (defined in Algorithm 12), the product is equal to the joint probability of the tag bigram and the entire input,

$$\begin{aligned} \alpha_{m-1}(k') \times s_m(k, k') \times \beta_m(k) &= p(Y_{m-1} = k', \mathbf{w}_{1:m-1}) \\ &\quad \times p(Y_m = k, w_m | Y_{m-1} = k') \\ &\quad \times p(\mathbf{w}_{m+1:M} | Y_m = k) \\ &= p(Y_{m-1} = k', Y_m = k, \mathbf{w}_{1:M}). \end{aligned} \quad [7.105]$$

Dividing by  $\alpha_{M+1}(\blacklozenge) = p(\mathbf{w}_{1:M})$  gives the desired probability,

$$\frac{\alpha_{m-1}(k') \times s_m(k, k') \times \beta_m(k)}{\alpha_{M+1}(\blacklozenge)} = \frac{p(Y_{m-1} = k', Y_m = k, \mathbf{w}_{1:M})}{p(\mathbf{w}_{1:M})} \quad [7.106]$$

$$= \Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}_{1:M}). \quad [7.107]$$

3859 The expected emission counts can be computed in a similar manner, using the product  
 3860  $\alpha_m(k) \times \beta_m(k)$ .

3861 **7.7.1 Linear dynamical systems**

3862 The forward-backward algorithm can be viewed as Bayesian state estimation in a discrete  
 3863 state space. In a continuous state space,  $\mathbf{y}_m \in \mathbb{R}^K$ , the equivalent algorithm is the **Kalman**  
 3864 **smoother**. It also computes marginals  $p(\mathbf{y}_m | \mathbf{x}_{1:M})$ , using a similar two-step algorithm  
 3865 of forward and backward passes. Instead of computing a trellis of values at each step, the  
 3866 Kalman smoother computes a probability density function  $q_{\mathbf{y}_m}(\mathbf{y}_m; \boldsymbol{\mu}_m, \Sigma_m)$ , character-  
 3867 ized by a mean  $\boldsymbol{\mu}_m$  and a covariance  $\Sigma_m$  around the latent state. Connections between the  
 3868 Kalman Smoother and the forward-backward algorithm are elucidated by Minka (1999)  
 3869 and Murphy (2012).

3870 **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):

$$p(y_m | \mathbf{y}_{-m}, \mathbf{w}_{1:M}) \propto p(w_m | y_m) p(y_m | \mathbf{y}_{-m}). \quad [7.108]$$

3871 Gibbs Sampling has been applied to unsupervised part-of-speech tagging by Goldwater  
 3872 and Griffiths (2007). **Beam sampling** is a more sophisticated sampling algorithm, which  
 3873 randomly draws entire sequences  $\mathbf{y}_{1:M}$ , rather than individual tags  $y_m$ ; this algorithm  
 3874 was applied to unsupervised part-of-speech tagging by Van Gael et al. (2009). Spectral  
 3875 learning (see § 5.5.2) can also be applied to sequence labeling. By factoring matrices of  
 3876 co-occurrence counts of word bigrams and trigrams (Song et al., 2010; Hsu et al., 2012), it  
 3877 is possible to obtain globally optimal estimates of the transition and emission parameters,  
 3878 under mild assumptions.

3879 **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.<sup>12</sup> Given these operators, we can denote a general-

---

<sup>12</sup>In a semiring, the addition and multiplication operators must both obey associativity, and multiplication must distribute across addition; the addition operator must be commutative; there must be additive and multiplicative identities  $\bar{0}$  and  $\bar{1}$ , such that  $a \oplus \bar{0} = a$  and  $a \otimes \bar{1} = a$ ; and there must be a multiplicative annihilator  $\bar{0}$ , such that  $a \otimes \bar{0} = \bar{0}$ .

ized Viterbi recurrence as,

$$v_m(k) = \bigoplus_{k' \in \mathcal{Y}} s_m(k, k') \otimes v_{m-1}(k'). \quad [7.109]$$

3880     Each recurrence that we have seen so far is a special case of this generalized Viterbi  
3881     recurrence:

- 3882     • In the max-product Viterbi recurrence over probabilities, the  $\oplus$  operation corre-  
3883         sponds to maximization, and the  $\otimes$  operation corresponds to multiplication.
- 3884     • In the forward recurrence over probabilities, the  $\oplus$  operation corresponds to addi-  
3885         tion, and the  $\otimes$  operation corresponds to multiplication.
- 3886     • In the max-product Viterbi recurrence over log-probabilities, the  $\oplus$  operation corre-  
3887         sponds to maximization, and the  $\otimes$  operation corresponds to addition.<sup>13</sup>
- 3888     • In the forward recurrence over log-probabilities, the  $\oplus$  operation corresponds to log-  
3889         addition,  $a \oplus b = \log(e^a + e^b)$ . The  $\otimes$  operation corresponds to addition.

3890     The mathematical abstraction offered by semiring notation can be applied to the soft-  
3891     ware implementations of these algorithms, yielding concise and modular implemen-  
3892     tations. The OPENFST library (Allauzen et al., 2007) is an example of a software package in  
3893     which the algorithms are parametrized by the choice of semiring.

## 3894 Exercises

- 3895     1. Consider the garden path sentence, *The old man the boat*. Given word-tag and tag-tag  
3896         features, what inequality in the weights must hold for the correct tag sequence to  
3897         outscore the garden path tag sequence for this example?
- 3898     2. Sketch out an algorithm for a variant of Viterbi that returns the top- $n$  label se-  
3899         quences. What is the time and space complexity of this algorithm?
- 3900     3. Show how to compute the marginal probability  $\Pr(y_{m-2} = k, y_m = k' \mid \mathbf{w}_{1:M})$ , in  
3901         terms of the forwards and backward variables, and the potentials  $s_n(y_n, y_{n-1})$ .
- 3902     4. Suppose you receive a stream of text, where some of tokens have been replaced at  
3903         random with *NOISE*. For example:
  - 3904             • Source: *I try all things, I achieve what I can*
  - 3905             • Message received: *I try NOISE NOISE, I NOISE what I NOISE*

---

<sup>13</sup>This is sometimes called the **tropical semiring**, in honor of the Brazilian mathematician Imre Simon.

3906 Assume you have access to a pre-trained bigram language model, which gives prob-  
3907 abilities  $p(w_m \mid w_{m-1})$ . These probabilities can be assumed to be non-zero for all  
3908 bigrams.

- 3909 a) Show how to use the Viterbi algorithm to try to recover the source by maxi-  
3910 mizing the bigram language model log-probability. Specifically, set the scores  
3911  $s_m(y_m, y_{m-1})$  so that the Viterbi algorithm selects a sequence of words that  
3912 maximizes the bigram language model log-probability, *while leaving the non-*  
3913 *noise tokens intact*. Your solution should not modify the logic of the Viterbi  
3914 algorithm, it should only set the scores  $s_m(y_m, y_{m-1})$ .
- 3915 b) An alternative solution is to iterate through the text from  $m \in \{1, 2, \dots, M\}$ ,  
3916 replacing each noise token with the word that maximizes  $P(w_m \mid w_{m-1})$  ac-  
3917 cording to the bigram language model. Given an upper bound on the expected  
3918 fraction of tokens for which the two approaches will disagree.
- 3919 5. Consider an RNN tagging model with a tanh activation function on the hidden  
3920 layer, and a hinge loss on the output. (The problem also works for the margin loss  
3921 and negative log-likelihood.) Suppose you initialize all parameters to zero: this  
3922 includes the word embeddings that make up  $\mathbf{x}$ , the transition matrix  $\Theta$ , the out-  
3923 put weights  $\beta$ , and the initial hidden state  $\mathbf{h}_0$ . Prove that for any data and for any  
3924 gradient-based learning algorithm, all parameters will be stuck at zero.  
3925 Extra credit: would a sigmoid activation function avoid this problem?

## 3926 Chapter 8

# 3927 Applications of sequence labeling

3928 Sequence labeling has applications throughout natural language processing. This chapter  
3929 focuses on part-of-speech tagging, morpho-syntactic attribute tagging, named entity  
3930 recognition, and tokenization. It also touches briefly on two applications to interactive  
3931 settings: dialogue act recognition and the detection of code-switching points between  
3932 languages.

### 3933 8.1 Part-of-speech tagging

3934 The **syntax** of a language is the set of principles under which sequences of words are  
3935 judged to be grammatically acceptable by fluent speakers. One of the most basic syntactic  
3936 concepts is the **part-of-speech** (POS), which refers to the syntactic role of each word in a  
3937 sentence. This concept was used informally in the previous chapter, and you may have  
3938 some intuitions from your own study of English. For example, in the sentence *We like*  
3939 *vegetarian sandwiches*, you may already know that *we* and *sandwiches* are nouns, *like* is a  
3940 verb, and *vegetarian* is an adjective. These labels depend on the context in which the word  
3941 appears: in *she eats like a vegetarian*, the word *like* is a preposition, and the word *vegetarian*  
3942 is a noun.

3943 Parts-of-speech can help to disentangle or explain various linguistic problems. Recall  
3944 Chomsky's proposed distinction in chapter 6:

- 3945 (8.1) Colorless green ideas sleep furiously.
- 3946 (8.2) \*Ideas colorless furiously green sleep.

3947 One difference between these two examples is that the first contains part-of-speech transitions  
3948 that are typical in English: adjective to adjective, adjective to noun, noun to verb, and verb  
3949 to adverb. The second example contains transitions that are unusual: noun to adjective  
3950 and adjective to verb. The ambiguity in a headline like,

3951 (8.3) Teacher Strikes Idle Children

3952 can also be explained in terms of parts of speech: in the interpretation that was likely  
 3953 intended, *strikes* is a noun and *idle* is a verb; in the alternative explanation, *strikes* is a verb  
 3954 and *idle* is an adjective.

3955 Part-of-speech tagging is often taken as a early step in a natural language processing  
 3956 pipeline. Indeed, parts-of-speech provide features that can be useful for many of the  
 3957 tasks that we will encounter later, such as parsing (chapter 10), coreference resolution  
 3958 (chapter 15), and relation extraction (chapter 17).

3959 **8.1.1 Parts-of-Speech**

3960 The **Universal Dependencies** project (UD) is an effort to create syntactically-annotated  
 3961 corpora across many languages, using a single annotation standard (Nivre et al., 2016). As  
 3962 part of this effort, they have designed a part-of-speech **tagset**, which is meant to capture  
 3963 word classes across as many languages as possible.<sup>1</sup> This section describes that inventory,  
 3964 giving rough definitions for each of tags, along with supporting examples.

3965 Part-of-speech tags are **morphosyntactic**, rather than **semantic**, categories. This means  
 3966 that they describe words in terms of how they pattern together and how they are inter-  
 3967 nally constructed (e.g., what suffixes and prefixes they include). For example, you may  
 3968 think of a noun as referring to objects or concepts, and verbs as referring to actions or  
 3969 events. But events can also be nouns:

3970 (8.4) ... the **howling** of the **shrieking** storm.

3971 Here *howling* and *shrieking* are events, but grammatically they act as a noun and adjective  
 3972 respectively.

3973 **8.1.1.1 The Universal Dependency part-of-speech tagset**

3974 The UD tagset is broken up into three groups: open class tags, closed class tags, and  
 3975 “others.”

3976 **Open class tags** Nearly all languages contain nouns, verbs, adjectives, and adverbs.<sup>2</sup>  
 3977 These are all **open word classes**, because new words can easily be added to them. The  
 3978 UD tagset includes two other tags that are open classes: proper nouns and interjections.

3979 • **Nouns** (UD tag: NOUN) tend to describe entities and concepts, e.g.,

---

<sup>1</sup>The 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.

<sup>2</sup>One prominent exception is Korean, which some linguists argue does not have adjectives Kim (2002).

3980 (8.5) **Toes** are scarce among veteran **blubber men**.

3981 In English, nouns tend to follow determiners and adjectives, and can play the subject  
3982 role in the sentence. They can be marked for the plural number by an -s suffix.

- 3983 • **Proper nouns** (PROPN) are tokens in names, which uniquely specify a given entity,

3984 (8.6) “**Moby Dick?**” shouted **Ahab**.

- 3985 • **Verbs** (VERB), according to the UD guidelines, “typically signal events and ac-  
3986 tions.” But they are also defined grammatically: they “can constitute a minimal  
3987 predicate in a clause, and govern the number and types of other constituents which  
3988 may occur in a clause.”<sup>3</sup>

3989 (8.7) “**Moby Dick?**” shouted Ahab.

3990 (8.8) Shall we **keep chasing** this murderous fish?

3991 English verbs tend to come in between the subject and some number of direct ob-  
3992 jects, depending on the verb. They can be marked for **tense** and **aspect** using suffixes  
3993 such as *-ed* and *-ing*. (These suffixes are an example of **inflectional morphology**,  
3994 which is discussed in more detail in § 9.1.4.)

- 3995 • **Adjectives** (ADJ) describe properties of entities,

3996 (8.9) Shall we keep chasing this **murderous** fish?

3997 (8.10) Toes are **scarce** among **veteran** blubber men.

3998 In the second example, *scarce* is a predicative adjective, linked to the subject by the  
3999 **copula verb** *are*. This means that In contrast, *murderous* and *veteran* are attribute  
4000 adjectives, modifying the noun phrase in which they are embedded.

- 4001 • **Adverbs** (ADV) describe properties of events, and may also modify adjectives or  
4002 other adverbs:

4003 (8.11) It is not down on any map; true places **never** are.

4004 (8.12) ... **treacherously** hidden beneath the loveliest tints of azure

4005 (8.13) Not drowned **entirely**, though.

- 4006 • **Interjections** (INTJ) are used in exclamations, e.g.,

4007 (8.14) **Aye aye!** it was that accursed white whale that razed me.

---

<sup>3</sup><http://universaldependencies.org/u/pos/VERB.html>

4008 **Closed class tags** Closed word classes rarely receive new members. They are sometimes  
 4009 referred to as **function words** — as opposed to **content words** — as they have little lexical  
 4010 meaning of their own, but rather, help to organize the components of the sentence.

- 4011 • **Adpositions** (ADP) describe the relationship between a complement (usually a noun  
 4012 phrase) and another unit in the sentence, typically a noun or verb phrase.

- 4013 (8.15) Toes are scarce **among** veteran blubber men.  
 4014 (8.16) It is not **down on** any map.  
 4015 (8.17) Give not thyself **up** then.

4016 As the examples show, English generally uses prepositions, which are adpositions  
 4017 that appear before their complement. (An exception is *ago*, as in, *we met three days*  
 4018 *ago*). Postpositions are used in other languages, such as Japanese and Turkish.

- 4019 • **Auxiliary verbs** (AUX) are a closed class of verbs that add information such as  
 4020 tense, aspect, person, and number.

- 4021 (8.18) **Shall** we keep chasing this murderous fish?  
 4022 (8.19) What the white whale was to Ahab, **has been** hinted.  
 4023 (8.20) Ahab **must** use tools.  
 4024 (8.21) Meditation and water **are** wedded forever.  
 4025 (8.22) Toes **are** scarce among veteran blubber men.

4026 The final example is a copula verb, which is also tagged as an auxiliary in the UD  
 4027 corpus.

- 4028 • **Coordinating conjunctions** (CCONJ) express relationships between two words or  
 4029 phrases, which play a parallel role:

- 4030 (8.23) Meditation **and** water are wedded forever.

- 4031 • **Subordinating conjunctions** (SCONJ) link two elements, making one syntactically  
 4032 subordinate to the other:

- 4033 (8.24) There is wisdom **that** is woe.

- 4034 • **Pronouns** (PRON) are words that substitute for nouns or noun phrases.

- 4035 (8.25) Be **it what it will**, I'll go to **it** laughing.  
 4036 (8.26) **I** try all things, **I** achieve **what I can**.

4037        The example includes the personal pronouns *I* and *it*, as well as the relative pronoun  
 4038        *what*. Other pronouns include *myself*, *somebody*, and *nothing*.

- 4039     • **Determiners** (DET) provide additional information about the nouns or noun phrases  
 4040        that they modify:

4041        (8.27) What **the** white whale was to Ahab, has been hinted.

4042        (8.28) It is not down on **any** map.

4043        (8.29) I try **all** things ...

4044        (8.30) Shall we keep chasing **this** murderous fish?

4045        Determiners include articles (*the*), possessive determiners (*their*), demonstratives  
 4046        (*this murderous fish*), and quantifiers (*any map*).

- 4047     • **Numerals** (NUM) are an infinite but closed class, which includes integers, fractions,  
 4048        and decimals, regardless of whether spelled out or written in numerical form.

4049        (8.31) How then can this **one** small heart beat.

4050        (8.32) I am going to put him down for the **three hundredth**.

- 4051     • **Particles** (PART) are a catch-all of function words that combine with other words or  
 4052        phrases, but do not meet the conditions of the other tags. In English, this includes  
 4053        the infinitival *to*, the possessive marker, and negation.

4054        (8.33) Better **to** sleep with a sober cannibal than a drunk Christian.

4055        (8.34) So man's insanity is heaven's sense

4056        (8.35) It is **not** down on any map

4057        As the second example shows, the possessive marker is not considered part of the  
 4058        same token as the word that it modifies, so that *man's* is split into two tokens. (Tok-  
 4059        enization is described in more detail in § 8.4.) A non-English example of a particle  
 4060        is the Japanese question marker *ka*, as in,<sup>4</sup>

4061        (8.36) *Sensei desu ka*

Teacher are ?

4062        Is she a teacher?

---

<sup>4</sup>In 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.

4063 **Other** The remaining UD tags include punctuation (PUN) and symbols (SYM). Punc-  
 4064 tuation is purely structural — e.g., commas, periods, colons — while symbols can carry  
 4065 content of their own. Examples of symbols include dollar and percentage symbols, math-  
 4066 ematical operators, emoticons, emojis, and internet addresses. A final catch-all tag is X,  
 4067 which is used for words that cannot be assigned another part-of-speech category. The X  
 4068 tag is also used in cases of **code switching** (between languages), described in § 8.5.

4069 **8.1.1.2 Other tagsets**

4070 Prior to the Universal Dependency treebank, part-of-speech tagging was performed us-  
 4071 ing language-specific tagsets. The dominant tagset for English was designed as part of  
 4072 the **Penn Treebank** (PTB), and it includes 45 tags — more than three times as many as  
 4073 the UD tagset. This granularity is reflected in distinctions between singular and plural  
 4074 nouns, verb tenses and aspects, possessive and non-possessive pronouns, comparative  
 4075 and superlative adjectives and adverbs (e.g., *faster, fastest*), and so on. The Brown corpus  
 4076 includes a tagset that is even more detailed, with 87 tags Francis (1964), including special  
 4077 tags for individual auxiliary verbs such as *be, do, and have*.

4078 Different languages make different distinctions, and so the PTB and Brown tagsets are  
 4079 not appropriate for a language such as Chinese, which does not mark the verb tense (Xia,  
 4080 2000); nor for Spanish, which marks every combination of person and number in the  
 4081 verb ending; nor for German, which marks the case of each noun phrase. Each of these  
 4082 languages requires more detail than English in some areas of the tagset, and less in other  
 4083 areas. The strategy of the Universal Dependencies corpus is to design a coarse-grained  
 4084 tagset to be used across all languages, and then to additionally annotate language-specific  
 4085 **morphosyntactic attributes**, such as number, tense, and case. The attribute tagging task  
 4086 is described in more detail in § 8.2.

4087 Social media such as Twitter have been shown to require tagsets of their own (Gimpel  
 4088 et al., 2011). Such corpora contain some tokens that are not equivalent to anything en-  
 4089 countered in a typical written corpus: e.g., emoticons, URLs, and hashtags. Social media  
 4090 also includes dialectal words like *gonna* ('going to', e.g. *We gonna be fine*) and *Ima* ('I'm  
 4091 going to', e.g., *Ima tell you one more time*), which can be analyzed either as non-standard  
 4092 orthography (making tokenization impossible), or as lexical items in their own right. In  
 4093 either case, it is clear that existing tags like NOUN and VERB cannot handle cases like *Ima*,  
 4094 which combine aspects of the noun and verb. Gimpel et al. (2011) therefore propose a new  
 4095 set of tags to deal with these cases.

4096 **8.1.2 Accurate part-of-speech tagging**

4097 Part-of-speech tagging is the problem of selecting the correct tag for each word in a sen-  
 4098 tence. Success is typically measured by accuracy on an annotated test set, which is simply  
 4099 the fraction of tokens that were tagged correctly.

## 4100 8.1.2.1 Baselines

4101 A simple baseline for part-of-speech tagging is to choose the most common tag for each  
4102 word. For example, in the Universal Dependencies treebank, the word *talk* appears 96  
4103 times, and 85 of those times it is labeled as a VERB: therefore, this baseline will always  
4104 predict VERB for this word. For words that do not appear in the training corpus, the base-  
4105 line simply guesses the most common tag overall, which is NOUN. In the Penn Treebank,  
4106 this simple baseline obtains accuracy above 92%. A more rigorous evaluation is the accu-  
4107 racy on **out-of-vocabulary words**, which are not seen in the training data. Tagging these  
4108 words correctly requires attention to the context and the word's internal structure.

## 4109 8.1.2.2 Contemporary approaches

4110 Conditional random fields and structured perceptron perform at or near the state-of-the-  
4111 art for part-of-speech tagging in English. For example, (Collins, 2002) achieved 97.1%  
4112 accuracy on the Penn Treebank, using a structured perceptron with the following base  
4113 features (originally introduced by Ratnaparkhi (1996)):

- 4114 • current word,  $w_m$
- 4115 • previous words,  $w_{m-1}, w_{m-2}$
- 4116 • next words,  $w_{m+1}, w_{m+2}$
- 4117 • previous tag,  $y_{m-1}$
- 4118 • previous two tags,  $(y_{m-1}, y_{m-2})$
- 4119 • for rare words:
  - 4120 – first  $k$  characters, up to  $k = 4$
  - 4121 – last  $k$  characters, up to  $k = 4$
  - 4122 – whether  $w_m$  contains a number, uppercase character, or hyphen.

4123 Similar results for the PTB data have been achieved using conditional random fields (CRFs;  
4124 Toutanova et al., 2003).

4125 More recent work has demonstrated the power of neural sequence models, such as the  
4126 **long short-term memory (LSTM)** (§ 7.6). Plank et al. (2016) apply a CRF and a bidirec-  
4127 tional LSTM to twenty-two languages in the UD corpus, achieving an average accuracy  
4128 of 94.3% for the CRF, and 96.5% with the bi-LSTM. Their neural model employs three  
4129 types of embeddings: fine-tuned word embeddings, which are updated during training;  
4130 pre-trained word embeddings, which are never updated, but which help to tag out-of-  
4131 vocabulary words; and character-based embeddings. The character-based embeddings  
4132 are computed by running an LSTM on the individual characters in each word, thereby  
4133 capturing common orthographic patterns such as prefixes, suffixes, and capitalization.  
4134 Extensive evaluations show that these additional embeddings are crucial to their model's  
4135 success.

word	PTB tag	UD tag	UD attributes
<i>The</i>	DT	DET	DEFINITE=DEF PRONTYPE=ART
<i>German</i>	JJ	ADJ	DEGREE=POS
<i>Expressionist</i>	NN	NOUN	NUMBER=SING
<i>movement</i>	NN	NOUN	NUMBER=SING
<i>was</i>	VBD	AUX	MOOD=IND NUMBER=SING PERSON=3 TENSE=PAST VERBFORM=FIN
<i>destroyed</i>	VBN	VERB	TENSE=PAST VERBFORM=PART VOICE=PASS
<i>as</i>	IN	ADP	
<i>a</i>	DT	DET	DEFINITE=IND PRONTYPE=ART
<i>result</i>	NN	NOUN	NUMBER=SING
.	.	PUNCT	

Figure 8.1: UD and PTB part-of-speech tags, and UD morphosyntactic attributes. Example selected from the UD 1.4 English corpus.

## 4136 8.2 Morphosyntactic Attributes

4137 There is considerably more to say about a word than whether it is a noun or a verb: in En-  
 4138 glish, verbs are distinguish by features such tense and aspect, nouns by number, adjectives  
 4139 by degree, and so on. These features are language-specific: other languages distinguish  
 4140 other features, such as **case** (the role of the noun with respect to the action of the sen-  
 4141 tence, which is marked in languages such as Latin and German<sup>5</sup>) and **evidentiality** (the  
 4142 source of information for the speaker’s statement, which is marked in languages such as  
 4143 Turkish). In the UD corpora, these attributes are annotated as feature-value pairs for each  
 4144 token.<sup>6</sup>

4145 An example is shown in Figure 8.1. The determiner *the* is marked with two attributes:  
 4146 PRONTYPE=ART, which indicates that it is an **article** (as opposed to another type of deter-

<sup>5</sup>Case is marked in English for some personal pronouns, e.g., *She saw her, They saw them*.

<sup>6</sup>The 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).

4147 miner or pronominal modifier), and DEFINITE=DEF, which indicates that it is a **definite**  
4148 **article** (referring to a specific, known entity). The verbs are each marked with several  
4149 attributes. The auxiliary verb *was* is third-person, singular, past tense, finite (conjugated),  
4150 and indicative (describing an event that has happened or is currently happenings); the  
4151 main verb *destroyed* is in participle form (so there is no additional person and number  
4152 information), past tense, and passive voice. Some, but not all, of these distinctions are  
4153 reflected in the PTB tags VBD (past-tense verb) and VBN (past participle).

4154 While there are thousands of papers on part-of-speech tagging, there is comparatively  
4155 little work on automatically labeling morphosyntactic attributes. Faruqui et al. (2016)  
4156 train a support vector machine classification model, using a minimal feature set that in-  
4157 cludes the word itself, its prefixes and suffixes, and type-level information listing all pos-  
4158 sible morphosyntactic attributes for each word and its neighbors. Mueller et al. (2013) use  
4159 a conditional random field (CRF), in which the tag space consists of all observed com-  
4160 binations of morphosyntactic attributes (e.g., the tag would be DEF+ART for the word  
4161 *the* in Figure 8.1). This massive tag space is managed by decomposing the feature space  
4162 over individual attributes, and pruning paths through the trellis. More recent work has  
4163 employed bidirectional LSTM sequence models. For example, Pinter et al. (2017) train  
4164 a bidirectional LSTM sequence model. The input layer and hidden vectors in the LSTM  
4165 are shared across attributes, but each attribute has its own output layer, culminating in  
4166 a softmax over all attribute values, e.g.  $y_t^{\text{NUMBER}} \in \{\text{SING}, \text{PLURAL}, \dots\}$ . They find that  
4167 character-level information is crucial, especially when the amount of labeled data is lim-  
4168 ited.

4169 Evaluation is performed by first computing recall and precision for each attribute.  
4170 These scores can then be averaged at either the type or token level to obtain micro- or  
4171 macro-*F*-MEASURE. Pinter et al. (2017) evaluate on 23 languages in the UD treebank,  
4172 reporting a median micro-*F*-MEASURE of 0.95. Performance is strongly correlated with the  
4173 size of the labeled dataset for each language, with a few outliers: for example, Chinese is  
4174 particularly difficult, because although the dataset is relatively large ( $10^5$  tokens in the UD  
4175 1.4 corpus), only 6% of tokens have any attributes, offering few useful labeled instances.

### 4176 8.3 Named Entity Recognition

4177 A classical problem in information extraction is to recognize and extract mentions of  
4178 **named entities** in text. In news documents, the core entity types are people, locations, and  
4179 organizations; more recently, the task has been extended to include amounts of money,  
4180 percentages, dates, and times. In item 8.37 (Figure 8.2), the named entities include: *The*  
4181 *U.S. Army*, an organization; *Atlanta*, a location; and *May 14, 1864*, a date. Named en-  
4182 tity recognition is also a key task in **biomedical natural language processing**, with entity  
4183 types including proteins, DNA, RNA, and cell lines (e.g., Collier et al., 2000; Ohta et al.,  
4184 2002). Figure 8.2 shows an example from the GENIA corpus of biomedical research ab-

- (8.37) *The U.S. Army captured Atlanta on May 14, 1864*  
 B-ORG I-ORG I-ORG O B-LOC O B-DATE I-DATE I-DATE I-DATE  
 (8.38) *Number of glucocorticoid receptors in lymphocytes and ...*  
 O O B-PROTEIN I-PROTEIN O B-CELLTYPE O ...

Figure 8.2: BIO notation for named entity recognition. Example (8.38) is drawn from the GENIA corpus of biomedical documents (Ohta et al., 2002).

4185 stracts.

4186 A standard approach to tagging named entity spans is to use discriminative sequence  
 4187 labeling methods such as conditional random fields. However, the named entity recogni-  
 4188 tion (NER) task would seem to be fundamentally different from sequence labeling tasks  
 4189 like part-of-speech tagging: rather than tagging each token, the goal is to recover *spans*  
 4190 of tokens, such as *The United States Army*.

4191 This is accomplished by the **BIO notation**, shown in Figure 8.2. Each token at the  
 4192 beginning of a name span is labeled with a B- prefix; each token within a name span is la-  
 4193 beled with an I- prefix. These prefixes are followed by a tag for the entity type, e.g. B-LOC  
 4194 for the beginning of a location, and I-PROTEIN for the inside of a protein name. Tokens  
 4195 that are not parts of name spans are labeled as O. From this representation, the entity  
 4196 name spans can be recovered unambiguously. This tagging scheme is also advantageous  
 4197 for learning: tokens at the beginning of name spans may have different properties than  
 4198 tokens within the name, and the learner can exploit this. This insight can be taken even  
 4199 further, with special labels for the last tokens of a name span, and for unique tokens in  
 4200 name spans, such as *Atlanta* in the example in Figure 8.2. This is called BILOU notation,  
 4201 and it can yield improvements in supervised named entity recognition (Ratinov and Roth,  
 4202 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*)

4203 Another source of features is to use **gazetteers**: lists of known entity names. For example,  
 4204 the U.S. Social Security Administration provides a list of tens of thousands of given names

- (1) 日文 章魚 怎麼 說?  
 Japanese octopus how say  
 How to say octopus in Japanese?
- (2) 日 文章 魚 怎麼 說?  
 Japan essay fish how say

Figure 8.3: An example of tokenization ambiguity in Chinese (Sproat et al., 1996)

4205 — more than could be observed in any annotated corpus. Tokens or spans that match an  
 4206 entry in a gazetteer can receive special features; this provides a way to incorporate hand-  
 4207 crafted resources such as name lists in a learning-driven framework.

4208 **Neural sequence labeling for NER** Current research has emphasized neural sequence  
 4209 labeling, using similar LSTM models to those employed in part-of-speech tagging (Ham-  
 4210 merton, 2003; Huang et al., 2015; Lample et al., 2016). The bidirectional LSTM-CRF (Fig-  
 4211 ure 7.4 in § 7.6) does particularly well on this task, due to its ability to model tag-to-tag  
 4212 dependencies. However, Strubell et al. (2017) show that **convolutional neural networks**  
 4213 can be equally accurate, with significant improvement in speed due to the efficiency of  
 4214 implementing ConvNets on **graphics processing units (GPUs)**. The key innovation in  
 4215 this work was the use of **dilated convolution**, which is described in more detail in § 3.4.

## 4216 8.4 Tokenization

4217 A basic problem for text analysis, first discussed in § 4.3.1, is to break the text into a se-  
 4218 quence of discrete tokens. For alphabetic languages such as English, deterministic scripts  
 4219 suffice to achieve accurate tokenization. However, in logographic writing systems such  
 4220 as Chinese script, words are typically composed of a small number of characters, with-  
 4221 out intervening whitespace. The tokenization must be determined by the reader, with  
 4222 the potential for occasional ambiguity, as shown in Figure 8.3. One approach is to match  
 4223 character sequences against a known dictionary (e.g., Sproat et al., 1996), using additional  
 4224 statistical information about word frequency. However, no dictionary is completely com-  
 4225 prehensive, and dictionary-based approaches can struggle with such out-of-vocabulary  
 4226 words.

4227 Chinese tokenization has therefore been approached as a supervised sequence label-  
 4228 ing problem. Xue et al. (2003) train a logistic regression classifier to make independent  
 4229 segmentation decisions while moving a sliding window across the document. A set of  
 4230 rules is then used to convert these individual classification decisions into an overall tok-  
 4231 enization of the input. However, these individual decisions may be globally suboptimal,  
 4232 motivating a structure prediction approach. Peng et al. (2004) train a conditional random

4233 field to predict labels of START or NONSTART on each character. More recent work has  
 4234 employed neural network architectures. For example, Chen et al. (2015) use an LSTM-  
 4235 CRF architecture, as described in § 7.6: they construct a trellis, in which each tag is scored  
 4236 according to the hidden state of an LSTM, and tag-tag transitions are scored according  
 4237 to learned transition weights. The best-scoring segmentation is then computed by the  
 4238 Viterbi algorithm.

4239 8.5 Code switching

4240 Multilingual speakers and writers do not restrict themselves to a single language. **Code**  
4241 **switching** is the phenomenon of switching between languages in speech and text (Auer,  
4242 2013; Poplack, 1980). Written code switching has become more common in online social  
4243 media, as in the following extract from Justin Trudeau's website:<sup>7</sup>

- 4244 (8.39) *Although everything written on this site est disponible en anglais  
is available in English  
and in French, my personal videos seront bilingues  
will be bilingual*

4246 Accurately analyzing such texts requires first determining which languages are being  
4247 used. Furthermore, quantitative analysis of code switching can provide insights on the  
4248 languages themselves and their relative social positions.

Code switching can be viewed as a sequence labeling problem, where the goal is to label each token as a candidate switch point. In the example above, the words *est*, *and*, and *seront* would be labeled as switch points. Solorio and Liu (2008) detect English-Spanish switch points using a supervised classifier, with features that include the word, its part-of-speech in each language (according to a supervised part-of-speech tagger), and the probabilities of the word and part-of-speech in each language. Nguyen and Dogruöz (2013) apply a conditional random field to the problem of detecting code switching between Turkish and Dutch.

Code switching is a special case of the more general problem of word level language identification, which Barman et al. (2014) address in the context of trilingual code switching between Bengali, English, and Hindi. They further observe an even more challenging phenomenon: intra-word code switching, such as the use of English suffixes with Bengali roots. They therefore mark each token as either (1) belonging to one of the three languages; (2) a mix of multiple languages; (3) “universal” (e.g., symbols, numbers, emoticons); or (4) undefined.

<sup>7</sup>As quoted in <http://blogues.lapresse.ca/lagace/2008/09/08/justin-trudeau-really-parfait-bilingue/>, accessed August 21, 2017.

Speaker	Dialogue Act	Utterance
A	YES-NO-QUESTION	<i>So do you go college right now?</i>
A	ABANDONED	<i>Are yo-</i>
B	YES-ANSWER	<i>Yeah,</i>
B	STATEMENT	<i>It's my last year [laughter].</i>
A	DECLARATIVE-QUESTION	<i>You're a, so you're a senior now.</i>
B	YES-ANSWER	<i>Yeah,</i>
B	STATEMENT	<i>I'm working on my projects trying to graduate [laughter]</i>
A	APPRECIATION	<i>Oh, good for you.</i>
B	BACKCHANNEL	<i>Yeah.</i>

Figure 8.4: An example of dialogue act labeling (Stolcke et al., 2000)

## 4264 8.6 Dialogue acts

4265 The sequence labeling problems that we have discussed so far have been over sequences  
 4266 of word tokens or characters (in the case of tokenization). However, sequence labeling  
 4267 can also be performed over higher-level units, such as **utterances**. **Dialogue acts** are la-  
 4268 bels over utterances in a dialogue, corresponding roughly to the speaker’s intention —  
 4269 the utterance’s **illocutionary force** (Austin, 1962). For example, an utterance may state a  
 4270 proposition (*it is not down on any map*), pose a question (*shall we keep chasing this murderous*  
 4271 *fish?*), or provide a response (*aye aye!*). Stolcke et al. (2000) describe how a set of 42 dia-  
 4272 logue acts were annotated for the 1,155 conversations in the Switchboard corpus (Godfrey  
 4273 et al., 1992).<sup>8</sup>

4274 An example is shown in Figure 8.4. The annotation is performed over UTTERANCES,  
 4275 with the possibility of multiple utterances per **conversational turn** (in cases such as inter-  
 4276 ruptions, an utterance may split over multiple turns). Some utterances are clauses (e.g., *So*  
 4277 *do you go to college right now?*), while others are single words (e.g., *yeah*). Stolcke et al. (2000)  
 4278 report that hidden Markov models (HMMs) achieve 96% accuracy on supervised utter-  
 4279 ance segmentation. The labels themselves reflect the conversational goals of the speaker:  
 4280 the utterance *yeah* functions as an answer in response to the question *you’re a senior now*,  
 4281 but in the final line of the excerpt, it is a **backchannel** (demonstrating comprehension).

4282 For task of dialogue act labeling, Stolcke et al. (2000) apply a hidden Markov model.  
 4283 The probability  $p(w_m | y_m)$  must generate the entire sequence of words in the utterance,  
 4284 and it is modeled as a trigram language model (§ 6.1). Stolcke et al. (2000) also account  
 4285 for acoustic features, which capture the **prosody** of each utterance — for example, tonal  
 4286 and rhythmic properties of speech, which can be used to distinguish dialogue acts such

<sup>8</sup>Dialogue 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.

4287 as questions and answers. These features are handled with an additional emission distri-  
4288 bution,  $p(a_m | y_m)$ , which is modeled with a probabilistic decision tree (Murphy, 2012).  
4289 While acoustic features yield small improvements overall, they play an important role in  
4290 distinguish questions from statements, and agreements from backchannels.

4291 Recurrent neural architectures for dialogue act labeling have been proposed by Kalch-  
4292 brenner and Blunsom (2013) and Ji et al. (2016), with strong empirical results. Both models  
4293 are recurrent at the utterance level, so that each complete utterance updates a hidden state.  
4294 The recurrent-convolutional network of Kalchbrenner and Blunsom (2013) uses convolu-  
4295 tion to obtain a representation of each individual utterance, while Ji et al. (2016) use a  
4296 second level of recurrence, over individual words. This enables their method to also func-  
4297 tion as a language model, giving probabilities over sequences of words in a document.

## 4298 Exercises

- 4299 1. [todo: exercises tk]

4300 **Chapter 9**

4301 **Formal language theory**

4302 We have now seen methods for learning to label individual words, vectors of word counts,  
4303 and sequences of words; we will soon proceed to more complex structural transfor-  
4304 mations. Most of these techniques could apply to counts or sequences from any discrete vo-  
4305 cabulary; there is nothing fundamentally linguistic about, say, a hidden Markov model.  
4306 This raises a basic question that this text has not yet considered: what is a language?

4307 This chapter will take the perspective of **formal language theory**, in which a language  
4308 is defined as a set of **strings**, each of which is a sequence of elements from a finite alphabet.  
4309 For interesting languages, there are an infinite number of strings that are in the language,  
4310 and an infinite number of strings that are not. For example:

- 4311 • the set of all even-length sequences from the alphabet  $\{a, b\}$ , e.g.,  $\{\emptyset, aa, ab, ba, bb, aaaa, aaab, \dots\}$ ;
- 4312 • the set of all sequences from the alphabet  $\{a, b\}$  that contain *aaa* as a substring, e.g.,  
4313  $\{aaa, aaaa, baaa, aaab, \dots\}$ ;
- 4314 • the set of all sequences of English words (drawn from a finite dictionary) that con-  
4315 tain at least one verb (a finite subset of the dictionary);
- 4316 • the `python` programming language.

4317 Formal language theory defines classes of languages and their computational prop-  
4318 erties. Of particular interest is the computational complexity of solving the **membership**  
4319 **problem** — determining whether a string is in a language. The chapter will focus on  
4320 three classes of formal languages: regular, context-free, and “mildly” context-sensitive  
4321 languages.

4322 A key insight of 20th century linguistics is that formal language theory can be usefully  
4323 applied to natural languages such as English, by designing formal languages that cap-  
4324 ture as many properties of the natural language as possible. For many such formalisms, a  
4325 useful linguistic analysis comes as a byproduct of solving the membership problem. The

4326 membership problem can be generalized to the problems of *scoring* strings for their ac-  
 4327 ceptability (as in language modeling), and of **transducing** one string into another (as in  
 4328 translation).

## 4329 9.1 Regular languages

4330 Sooner or later, most computer scientists will write a **regular expression**. If you have,  
 4331 then you have defined a **regular language**, which is any language that can be defined by  
 4332 a regular expression. Formally, a regular expression can include the following elements:

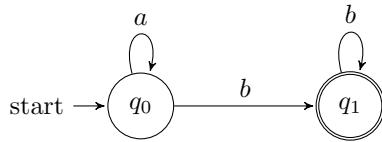
- 4333 • A **literal character** drawn from some finite alphabet  $\Sigma$ .
- 4334 • The **empty string**  $\epsilon$ .
- 4335 • The concatenation of two regular expressions  $RS$ , where  $R$  and  $S$  are both regular  
 4336 expressions. The resulting expression accepts any string that can be decomposed  
 4337  $x = yz$ , where  $y$  is accepted by  $R$  and  $z$  is accepted by  $S$ .
- 4338 • The alternation  $R \mid S$ , where  $R$  and  $S$  are both regular expressions. The resulting  
 4339 expression accepts a string  $x$  if it is accepted by  $R$  or it is accepted by  $S$ .
- 4340 • The **Kleene star**  $R^*$ , which accepts any string  $x$  that can be decomposed into a se-  
 4341 quence of strings which are all accepted by  $R$ .
- 4342 • Parenthesization ( $(R)$ ), which is used to limit the scope of the concatenation, alterna-  
 4343 tion, and Kleene star operators.

4344 Here are some example regular expressions:

- 4345 • The set of all even length strings on the alphabet  $\{a, b\}$ :  $((aa)|(ab)|(ba)|(bb))^*$
- 4346 • The set of all sequences of the alphabet  $\{a, b\}$  that contain  $aaa$  as a substring:  $(a|b)^*aaa(a|b)^*$
- 4347 • The set of all sequences of English words that contain at least one verb:  $W^*VW^*$ ,  
 4348 where  $W$  is an alternation between all words in the dictionary, and  $V$  is an alterna-  
 4349 tion between all verbs ( $V \subseteq W$ ).

4350 This list does not include a regular expression for the Python programming language,  
 4351 because this language is not regular — there is no regular expression that can capture its  
 4352 syntax. We will discuss why towards the end of this section.

4353 Regular languages are **closed** under union, intersection, and concatenation. This means,  
 4354 for example, that if two languages  $L_1$  and  $L_2$  are regular, then so are the languages  $L_1 \cup L_2$ ,  
 4355  $L_1 \cap L_2$ , and the language of strings that can be decomposed as  $s = tu$ , with  $s \in L_1$  and  
 4356  $t \in L_2$ . Regular languages are also closed under negation: if  $L$  is regular, then so is the  
 4357 language  $\bar{L} = \{s \notin L\}$ .

Figure 9.1: State diagram for the finite state acceptor  $M_1$ .4358 **9.1.1 Finite state acceptors**

4359 A regular expression defines a regular language, but does not give an algorithm for de-  
 4360 termining whether a string is in the language that it defines. **Finite state automata** are  
 4361 theoretical models of computation on regular languages, which involve transitions be-  
 4362 tween a finite number of states. The most basic type of finite state automaton is the **finite**  
 4363 **state acceptor (FSA)**, which describes the computation involved in testing if a string is  
 4364 a member of a language. Formally, a finite state acceptor is a tuple  $M = (Q, \Sigma, q_0, F, \delta)$ ,  
 4365 consisting of:

- 4366 • a finite alphabet  $\Sigma$  of input symbols;
- 4367 • a finite set of states  $Q = \{q_0, q_1, \dots, q_n\}$ ;
- 4368 • a start state  $q_0 \in Q$ ;
- 4369 • a set of final states  $F \subseteq Q$ ;
- 4370 • a transition function  $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$ . The transition function maps from a  
 4371 state and an input symbol (or empty string  $\epsilon$ ) to a *set* of possible resulting states.

4372 A **path** in  $M$  is a sequence of transitions,  $\pi = t_1, t_2, \dots, t_N$ , where each  $t_i$  traverses an  
 4373 arc in the transition function  $\delta$ . The finite state acceptor  $M$  accepts a string  $\omega$  if there is  
 4374 a **accepting path**, in which the initial transition  $t_1$  begins at the start state  $q_0$ , the final  
 4375 transition  $t_N$  terminates in a final state in  $Q$ , and the entire input  $\omega$  is consumed.

4376 **9.1.1.1 Example**

Consider the following FSA,  $M_1$ .

$$\Sigma = \{a, b\} \quad [9.1]$$

$$Q = \{q_0, q_1\} \quad [9.2]$$

$$F = \{q_1\} \quad [9.3]$$

$$\delta = \{(q_0, a) \rightarrow q_0, (q_0, b) \rightarrow q_1, (q_1, b) \rightarrow q_1\}. \quad [9.4]$$

4377 This FSA defines a language over an alphabet of two symbols,  $a$  and  $b$ . The transition  
 4378 function  $\delta$  is written as a set of arcs:  $(q_0, a) \rightarrow q_0$  says that if the machine is in state

4379  $q_0$  and reads symbol  $a$ , it stays in  $q_0$ . Figure 9.1 provides a graphical representation of  
 4380  $M_1$ . Because each pair of initial state and symbol has at most one resulting state,  $M_1$  is  
 4381 **deterministic**: each string  $\omega$  induces at most one accepting path. Note that there are no  
 4382 transitions for the symbol  $a$  in state  $q_1$ ; if  $a$  is encountered in  $q_1$ , then the acceptor is stuck,  
 4383 and the input string is rejected.

4384 What strings does  $M_1$  accept? The start state is  $q_0$ , and we have to get to  $q_1$ , since this  
 4385 is the only final state. Any number of  $a$  symbols can be consumed in  $q_0$ , but a  $b$  symbol is  
 4386 required to transition to  $q_1$ . Once there, any number of  $b$  symbols can be consumed, but  
 4387 an  $a$  symbol cannot. So the regular expression corresponding to the language defined by  
 4388  $M_1$  is  $a^*bb^*$ .

#### 4389 9.1.1.2 Computational properties of finite state acceptors

4390 The key computational question for finite state acceptors is: how fast can we determine  
 4391 whether a string is accepted? For deterministic FSAs, this computation can be performed  
 4392 by Dijkstra's algorithm, with time complexity  $\mathcal{O}(V \log V + E)$ , where  $V$  is the number of  
 4393 vertices in the FSA, and  $E$  is the number of edges (Cormen et al., 2009). Non-deterministic  
 4394 FSAs (NFSAs) can include multiple transitions from a given symbol and state. Any NSFA  
 4395 can be converted into a deterministic FSA, but the resulting automaton may have a num-  
 4396 ber of states that is exponential in the number of size of the original NFSFA (Mohri et al.,  
 4397 2002).

#### 4398 9.1.2 Morphology as a regular language

4399 Many words have internal structure, such as prefixes and suffixes that shape their mean-  
 4400 ing. The study of word-internal structure is the domain of **morphology**, of which there  
 4401 are two main types:

- 4402 • **Derivational morphology** describes the use of affixes to convert a word from one  
 4403 grammatical category to another (e.g., from the noun *grace* to the adjective *graceful*),  
 4404 or to change the meaning of the word (e.g., from *grace* to *disgrace*).
- 4405 • **Inflectional morphology** describes the addition of details such as gender, number,  
 4406 person, and tense (e.g., the *-ed* suffix for past tense in English).

4407 Morphology is a rich topic in linguistics, deserving of a course in its own right.<sup>1</sup> The  
 4408 focus here will be on the use of finite state automata for morphological analysis. The

---

<sup>1</sup>A good starting point would be a chapter from a linguistics textbook (e.g., Akmajian et al., 2010; Bender, 2013). A key simplification in this chapter is the focus on affixes at the sole method of derivation and inflection. English makes use of affixes, but also incorporates **apophony**, such as the inflection of *foot* to *feet*. Semitic languages like Arabic and Hebrew feature a template-based system of morphology, in which roots are triples of consonants (e.g., *ktb*), and words are created by adding vowels: *kataba* (Arabic: he wrote), *kutub* (books), *maktab* (desk). For more detail on morphology, see texts from Haspelmath and Sims (2013) and Lieber (2015).

4409 current section deals with derivational morphology; inflectional morphology is discussed  
4410 in § 9.1.4.3.

4411 Suppose that we want to write a program that accepts only those words that are con-  
4412 structed in accordance with the rules of English derivational morphology:

- 4413 (9.1) grace, graceful, gracefully, \*gracelyful  
4414 (9.2) disgrace, \*ungrace, disgraceful, disgracefully  
4415 (9.3) allure, \*allureful, alluring, alluringly  
4416 (9.4) fairness, unfair, \*disfair, fairly

4417 (Recall that the asterisk indicates that a linguistic example is judged unacceptable by flu-  
4418 ent speakers of a language.) These examples cover only a tiny corner of English deriva-  
4419 tional morphology, but a number of things stand out. The suffix *-ful* converts the nouns  
4420 *grace* and *disgrace* into adjectives, and the suffix *-ly* converts adjectives into adverbs. These  
4421 suffixes must be applied in the correct order, as shown by the unacceptability of *\*grace-*  
4422 *lyful*. The *-ful* suffix works for only some words, as shown by the use of *alluring* as the  
4423 adjectival form of *allure*. Other changes are made with prefixes, such as the derivation  
4424 of *disgrace* from *grace*, which roughly corresponds to a negation; however, *fair* is negated  
4425 with the *un-* prefix instead. Finally, while the first three examples suggest that the direc-  
4426 tion of derivation is noun → adjective → adverb, the example of *fair* suggests that the  
4427 adjective can also be the base form, with the *-ness* suffix performing the conversion to a  
4428 noun.

4429 Can we build a computer program that accepts only well-formed English words, and  
4430 rejects all others? This might at first seem trivial to solve with a brute-force attack: simply  
4431 make a dictionary of all valid English words. But such an approach fails to account for  
4432 morphological **productivity** — the applicability of existing morphological rules to new  
4433 words and names, such as *Trump* to *Trumpy* and *Trumpkin*, and *Clinton* to *Clintonian* and  
4434 *Clintonite*. We need an approach that represents morphological rules explicitly, and for  
4435 this we will try a finite state acceptor.

4436 The dictionary approach can be implemented as a finite state acceptor, with the vo-  
4437 cabulary  $\Sigma$  equal to the vocabulary of English, and a transition from the start state to the  
4438 accepting state for each word. But this would of course fail to generalize beyond the origi-  
4439 nal vocabulary, and would not capture anything about the **morphotactic** rules that govern  
4440 derivations from new words. The first step towards a more general approach is shown in  
4441 Figure 9.2, which is the state diagram for a finite state acceptor in which the vocabulary  
4442 consists of **morphemes**, which include **stems** (e.g., *grace*, *allure*) and **affixes** (e.g., *dis-*, *-ing*,  
4443 *-ly*). This finite state acceptor consists of a set of paths leading away from the start state,  
4444 with derivational affixes added along the path. Except for  $q_{\text{neg}}$ , the states on these paths  
4445 are all final, so the FSA will accept *disgrace*, *disgraceful*, and *disgracefully*, but not *dis-*.

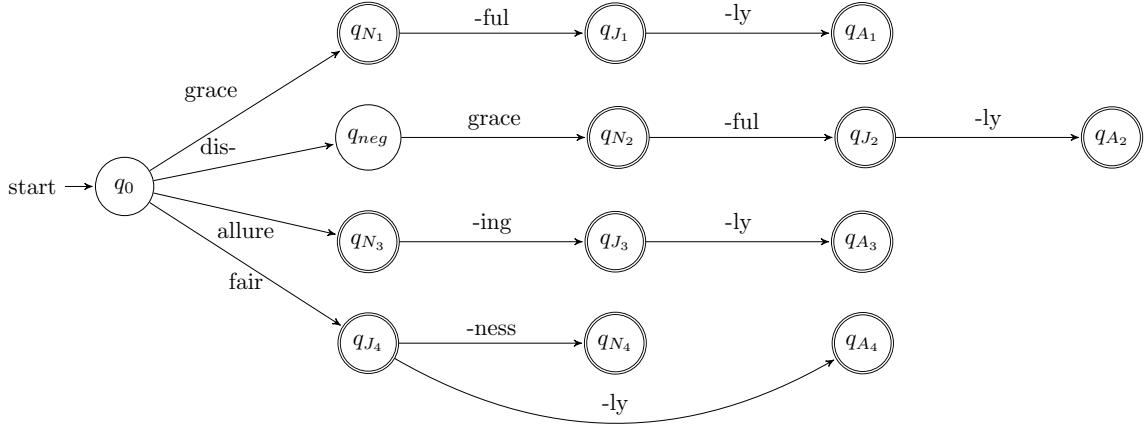


Figure 9.2: A finite state acceptor for a fragment of English derivational morphology. Each path represents possible derivations from a single root form.

4446     This FSA can be **minimized** to the form shown in Figure 9.3, which makes the general-  
 4447     ility of the finite state approach more apparent. For example, the transition from  $q_0$  to  
 4448      $q_{J_2}$  can be made to accept not only *fair* but any single-morpheme (**monomorphemic**) ad-  
 4449     jective that takes *-ness* and *-ly* as suffixes. In this way, the finite state acceptor can easily  
 4450     be extended: as new word stems are added to the vocabulary, their derived forms will be  
 4451     accepted automatically. Of course, this FSA would still need to be extended considerably  
 4452     to cover even this small fragment of English morphology. As shown by cases like *music*  
 4453     → *musical*, *athlete* → *athletic*, English includes several classes of nouns, each with its own  
 4454     rules for derivation.

4455     The FSAs shown in Figure 9.2 and 9.3 accept *allureing*, not *alluring*. This reflects a dis-  
 4456     tinction between morphology — the question of which morphemes to use, and in what  
 4457     order — and **orthography** — the question of how the morphemes are rendered in written  
 4458     language. Just as orthography requires dropping the *e* preceding the *-ing* suffix, **phonol-**  
 4459     **ogy** imposes a related set of constraints on how words are rendered in speech. As we will  
 4460     see soon, these issues are handled through **finite state transducers**, which are finite state  
 4461     automata that take inputs and produce outputs.

### 4462     9.1.3 Weighted finite state acceptors

4463     According to the FSA treatment of morphology, every word is either in or out of the lan-  
 4464     guage, with no wiggle room. Perhaps you agree that *musicky* and *fishful* are not valid  
 4465     English words; but if forced to choose, you probably find *a fishful stew* or *a musicky trib-*  
 4466     *ute* preferable to *behaving disgracelyful*. Rather than asking whether a word is acceptable,  
 4467     we might like to ask how acceptable it is. Aronoff (1976, page 36) puts it another way:

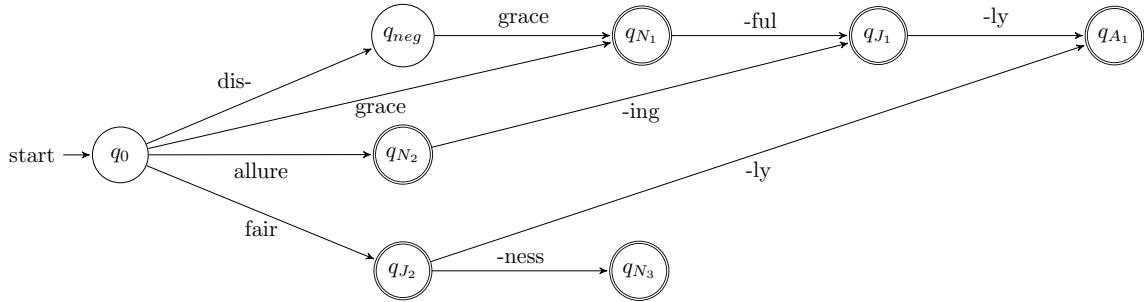


Figure 9.3: Minimization of the finite state acceptor shown in Figure 9.2.

4468 “Though many things are possible in morphology, some are more possible than others.”  
 4469 But finite state acceptors give no way to express preferences among technically valid  
 4470 choices.

4471 **Weighted finite state acceptors (WFSAs)** are generalizations of FSAs, in which each  
 4472 accepting path is assigned a score, computed from the transitions, the initial state, and the  
 4473 final state. Formally, a weighted finite state acceptor  $M = (Q, \Sigma, \lambda, \rho, \delta)$  consists of:

- 4474 • a finite set of states  $Q = \{q_0, q_1, \dots, q_n\}$ ;
- 4475 • a finite alphabet  $\Sigma$  of input symbols;
- 4476 • an initial weight function,  $\lambda : Q \mapsto \mathbb{R}$ ;
- 4477 • a final weight function  $\rho : Q \mapsto \mathbb{R}$ ;
- 4478 • a transition function  $\delta : Q \times \Sigma \times Q \mapsto \mathbb{R}$ .

4479 WFSAs depart from the FSA formalism in three ways: every state can be an initial  
 4480 state, with score  $\lambda(q)$ ; every state can be an accepting state, with score  $\rho(q)$ ; transitions are  
 4481 possible between any pair of states on any input, with a score  $\delta(q_i, \omega, q_j)$ . Nonetheless,  
 4482 FSAs can be viewed as a special case: for any FSA  $M$  we can build an equivalent WFSAs  
 4483 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  
 4484 transitions  $\{(q_1, \omega) \rightarrow q_2\}$  that are not permitted by the transition function of  $M$ .

4485 The total score for any path  $\pi = t_1, t_2, \dots, t_N$  is equal to the sum of these scores,

$$d(\pi) = \lambda(\text{from-state}(t_1)) + \sum_n^N \delta(t_n) + \rho(\text{to-state}(t_N)). \quad [9.5]$$

4486 A **shortest-path algorithm** is used to find the minimum-cost path through a WFSAs for  
 4487 string  $\omega$ , with time complexity  $\mathcal{O}(E + V \log V)$ , where  $E$  is the number of edges and  $V$  is  
 4488 the number of vertices (Cormen et al., 2009).<sup>2</sup>

<sup>2</sup>Shortest-path algorithms find the path with the minimum cost. In many cases, the path weights are log

4489 **9.1.3.1 N-gram language models as WFSAs**

4490 In **n-gram language models** (see § 6.1), the probability of a sequence of tokens  $w_1, w_2, \dots, w_M$   
 4491 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:

$$\begin{aligned} \delta(q_i, \omega, q_j) &= \begin{cases} \log \Pr(w_m = j | w_{m-1} = i), & \omega = j \\ -\infty, & \omega \neq j \end{cases} \\ \lambda(q_i) &= \log \Pr(w_1 = i | w_0 = \square) \\ \rho(q_i) &= \log \Pr(w_{M+1} = \blacksquare | w_M = i). \end{aligned}$$

4492 The transition function is designed to ensure that the context is recorded accurately:  
 4493 we can move to state  $j$  on input  $\omega$  only if  $\omega = j$ ; otherwise, transitioning to state  $j$  is  
 4494 forbidden by the weight of  $-\infty$ . The initial weight function  $\lambda(q_i)$  is the log probability of  
 4495 receiving  $i$  as the first token, and the final weight function  $\rho(q_i)$  is the log probability of  
 4496 receiving an “end-of-string” token after observing  $w_M = i$ .

4497 **9.1.3.2 \*Semiring weighted finite state acceptors**

4498 The  $n$ -gram language model WFSA is deterministic: each input has exactly one accepting  
 4499 path, for which the WFSA computes a score. In non-deterministic WFSAs, a given input

---

probabilities, so we want the path with the maximum score, which can be accomplished by making each local score into a *negative* log-probability. The remainder of this section will refer to **best-path algorithms**, which are assumed to “do the right thing.”

4500 may have multiple accepting paths. In some applications, the score for the input is ag-  
 4501 gregated across all such paths. Such aggregate scores can be computed by generalizing  
 4502 WFSAs with **semiring notation**, first introduced in § 7.7.3.

4503 Let  $d(\pi)$  represent the total score for path  $\pi = t_1, t_2, \dots, t_N$ , which is computed as,

$$d(\pi) = \lambda(\text{from-state}(t_1)) \otimes \delta(t_1) \otimes \delta(t_2) \otimes \dots \otimes \delta(t_N) \otimes \rho(\text{to-state}(t_N)). \quad [9.8]$$

4504 This is a generalization of Equation 9.5 to semiring notation, using the semiring multipli-  
 4505 cation operator  $\otimes$  in place of addition.

4506 Now let  $s(\omega)$  represent the total score for all paths  $\Pi(\omega)$  that consume input  $\omega$ ,

$$s(\omega) = \bigoplus_{\pi \in \Pi(\omega)} d(\pi). \quad [9.9]$$

4507 Here, semiring addition ( $\oplus$ ) is used to combine the scores of multiple paths.

4508 The generalization to semirings covers a number of useful special cases. In the log-  
 4509 probability semiring, multiplication is defined as  $\log p(x) \otimes \log p(y) = \log p(x) + \log p(y)$ ,  
 4510 and addition is defined as  $\log p(x) \oplus \log p(y) = \log(p(x) + p(y))$ . Thus,  $s(\omega)$  represents  
 4511 the log-probability of accepting input  $\omega$ , marginalizing over all paths  $\pi \in \Pi(\omega)$ . In the  
 4512 **boolean semiring**, the  $\otimes$  operator is logical conjunction, and the  $\oplus$  operator is logical  
 4513 disjunction. This reduces to the special case of unweighted finite state acceptors, where  
 4514 the score  $s(\omega)$  is a boolean indicating whether there exists any accepting path for  $\omega$ . In  
 4515 the **tropical semiring**, the  $\oplus$  operator is a maximum, so the resulting score is the score of  
 4516 the best-scoring path through the WFSAs. The OpenFST toolkit uses semirings and poly-  
 4517 morphism to implement general algorithms for weighted finite state automata (Allauzen  
 4518 et al., 2007).

### 4519 9.1.3.3 \*Interpolated $n$ -gram language models

4520 Recall from § 6.2.3 that an **interpolated  $n$ -gram language model** combines the probabili-  
 4521 ties from multiple  $n$ -gram models. For example, an interpolated bigram language model  
 4522 computes probability,

$$\hat{p}(w_m | w_{m-1}) = \lambda_1 p_1(w_m) + \lambda_2 p_2(w_m | w_{m-1}), \quad [9.10]$$

4523 with  $\hat{p}$  indicating the interpolated probability,  $p_2$  indicating the bigram probability, and  
 4524  $p_1$  indicating the unigram probability. We set  $\lambda_2 = (1 - \lambda_1)$  so that the probabilities sum  
 4525 to one.

4526 Interpolated bigram language models can be implemented using a non-deterministic  
 4527 WFSAs (Knight and May, 2009). The basic idea is shown in Figure 9.4. In an interpolated  
 4528 bigram language model, there is one state for each element in the vocabulary — in this

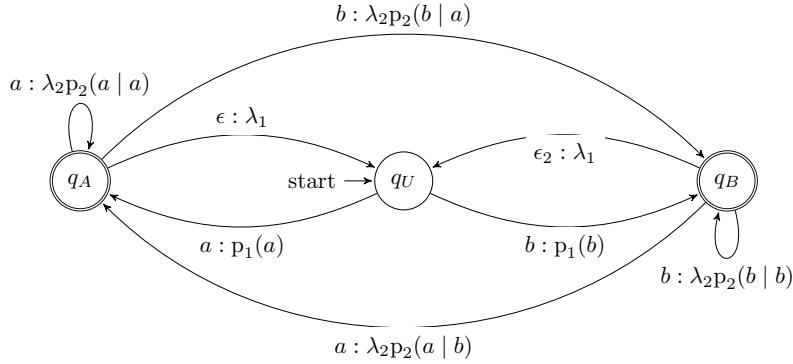


Figure 9.4: WFSA implementing an interpolated bigram/unigram language model, on the alphabet  $\Sigma = \{a, b\}$ . For simplicity, the WFSA is constrained to force the first token to be generated from the unigram model, and does not model the emission of the end-of-sequence token.

4529 case, the states  $q_A$  and  $q_B$  — which capture the contextual conditioning in the bigram  
 4530 probabilities. To model unigram probabilities, there is an additional state  $q_U$ , which “for-  
 4531 gets” the context. Transitions out of  $q_U$  involve unigram probabilities,  $p_1(a)$  and  $p_2(b)$ ;  
 4532 transitions into  $q_U$  emit the empty symbol  $\epsilon$ , and have probability  $\lambda_1$ , reflecting the inter-  
 4533 polation weight for the unigram model. The interpolation weight for the bigram model is  
 4534 included in the weight of the transition  $q_A \rightarrow q_B$ .

4535 The epsilon transitions into  $q_U$  make this WFSA non-deterministic. Consider the score  
 4536 for the sequence  $(a, b, b)$ . The initial state is  $q_U$ , so the symbol  $a$  is generated with score  
 4537  $p_1(a)$ <sup>3</sup> Next, we can generate  $b$  from the unigram model by taking the transition  $q_A \rightarrow q_B$ ,  
 4538 with score  $\lambda_2 p_2(b | a)$ . Alternatively, we can take a transition back to  $q_U$  with score  $\lambda_1$ ,  
 4539 and then emit  $b$  from the unigram model with score  $p_1(b)$ . To generate the final  $b$  token,  
 4540 we face the same choice: emit it directly from the self-transition to  $q_B$ , or transition to  $q_U$   
 4541 first.

The total score for the sequence  $(a, b, b)$  is the semiring sum over all accepting paths,

$$\begin{aligned}
 s(a, b, b) &= (p_1(a) \otimes \lambda_2 p_2(b | a) \otimes \lambda_2 p_2(b | b)) \\
 &\oplus (p_1(a) \otimes \lambda_1 \otimes p_1(b) \otimes \lambda_2 p_2(b | b)) \\
 &\oplus (p_1(a) \otimes \lambda_2 p_2(b | a) \otimes p_1(b) \otimes p_1(b)) \\
 &\oplus (p_1(a) \otimes \lambda_1 \otimes p_1(b) \otimes p_1(b) \otimes p_1(b)). \tag{[9.11]}
 \end{aligned}$$

4542 Each line in Equation 9.11 represents the probability of a specific path through the WFSA.  
 4543 In the probability semiring,  $\otimes$  is multiplication, so that each path is the product of each

<sup>3</sup>We 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.

4544 transition weight, which are themselves probabilities. The  $\oplus$  operator is addition, so that  
 4545 the total score is the sum of the scores (probabilities) for each path. This corresponds to  
 4546 the probability under the interpolated bigram language model.

4547 **9.1.4 Finite state transducers**

4548 Finite state acceptors can determine whether a string is in a regular language, and weighted  
 4549 finite state acceptors can compute a score for every string over a given alphabet. **Finite**  
 4550 **state transducers** (FSTs) extend the formalism further, by adding an output symbol to each  
 4551 transition. Formally, a finite state transducer is a tuple  $T = (Q, \Sigma, \Omega, \lambda, \rho, \delta)$ , with  $\Omega$  repre-  
 4552 senting an output vocabulary and the transition function  $\delta : Q \times (\Sigma \cup \epsilon) \times (\Omega \cup \epsilon) \times Q \rightarrow \mathbb{R}$   
 4553 mapping from states, input symbols, and output symbols to states. The remaining ele-  
 4554 ments ( $Q, \Sigma, \lambda, \rho$ ) are identical to their definition in weighted finite state acceptors (§ 9.1.3).  
 4555 Thus, each path through the FST  $T$  transduces the input string into an output.

4556 **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,

$$\delta(q, a, a, q) = \delta(q, b, b, q) = 0 \quad [9.12]$$

$$\delta(q, a, b, q) = \delta(q, b, a, q) = 1 \quad [9.13]$$

$$\delta(q, a, \epsilon, q) = \delta(q, b, \epsilon, q) = 1 \quad [9.14]$$

$$\delta(q, \epsilon, a, q) = \delta(q, \epsilon, b, q) = 1. \quad [9.15]$$

4557 The state diagram is shown in Figure 9.5.

4558 For a given string pair, there are multiple paths through the transducer: the best-  
 4559 scoring path from *dessert* to *desert* involves a single deletion, for a total score of 1; the  
 4560 worst-scoring path involves seven deletions and six additions, for a score of 13.

4561 **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

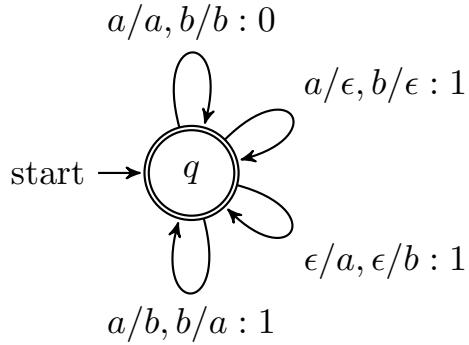


Figure 9.5: State diagram for the Levenshtein edit distance finite state transducer. The label  $x/y : c$  indicates a cost of  $c$  for a transition with input  $x$  and output  $y$ .

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]

4562 The final two lines appear to conflict; they are meant to be interpreted as an instruction  
 4563 to remove a terminal *-s* unless it is part of an *-ss* ending. A state diagram to handle just  
 4564 these final two lines is shown in Figure 9.6. Make sure you understand how this finite  
 4565 state transducer handles *cats*, *steps*, *bass*, and *basses*.

#### 4566 9.1.4.3 Inflectional morphology

4567 In **inflectional morphology**, word **lemmas** are modified to add grammatical information  
 4568 such as tense, number, and case. For example, many English nouns are pluralized by the  
 4569 suffix *-s*, and many verbs are converted to past tense by the suffix *-ed*. English's inflectional  
 4570 morphology is considerably simpler than many of the world's languages. For example,  
 4571 Romance languages (derived from Latin) feature complex systems of verb suffixes which  
 4572 must agree with the person and number of the verb, as shown in Table 9.1.

4573 The task of **morphological analysis** is to read a form like *canto*, and output an analysis  
 4574 like CANTAR+VERB+PRESIND+1P+SING, where +PRESIND describes the tense as present  
 4575 indicative, +1P indicates the first-person, and +SING indicates the singular number. The  
 4576 task of **morphological generation** is the reverse, going from CANTAR+VERB+PRESIND+1P+SING  
 4577 to *canto*. Finite state transducers are an attractive solution, because they can solve both  
 4578 problems with a single model (Beesley and Karttunen, 2003). As an example, Figure 9.7  
 4579 shows a fragment of a finite state transducer for Spanish inflectional morphology. The

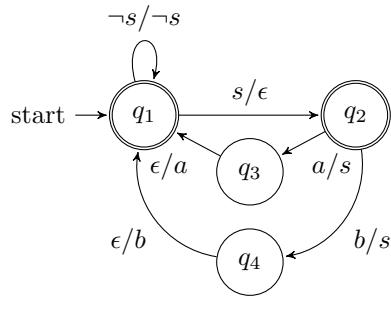


Figure 9.6: State diagram for final two lines of step 1a of the Porter stemming diagram. States  $q_3$  and  $q_4$  “remember” the observations  $a$  and  $b$  respectively; the ellipsis  $\dots$  represents additional states for each symbol in the input alphabet. The notation  $\neg s / \neg s$  is not part of the FST formalism; it is a shorthand to indicate a set of self-transition arcs for every input/output symbol except  $s$ .

infinitive	cantar (to sing)	comer (to eat)	vivir (to live)
yo (1st singular)	canto	como	vivo
tu (2nd singular)	cantas	comes	vives
él, ella, usted (3rd singular)	canta	come	vive
nosotros (1st plural)	cantamos	comemos	vivimos
vosotros (2nd plural, informal)	cantáis	coméis	vívís
ellos, ellas (3rd plural); ustedes (2nd plural)	cantan	comen	viven

Table 9.1: Spanish verb inflections for the present indicative tense. Each row represents a person and number, and each column is a regular example from a class of verbs, as indicated by the ending of the infinitive form.

4580 input vocabulary  $\Sigma$  corresponds to the set of letters used in Spanish spelling, and the out-  
 4581 put vocabulary  $\Omega$  corresponds to these same letters, plus the vocabulary of morphological  
 4582 features (e.g., +SING, +VERB). In Figure 9.7, there are two paths that take *canto* as input,  
 4583 corresponding to the verb and noun meanings; the choice between these paths could be  
 4584 guided by a part-of-speech tagger. By **inversion**, the inputs and outputs for each trans-  
 4585ition are switched, resulting in a finite state generator, capable of producing the correct  
 4586 **surface form** for any morphological analysis.

4587 Finite state morphological analyzers and other unweighted transducers can be de-  
 4588 signed by hand. The designer’s goal is to avoid **overgeneration** — accepting strings or  
 4589 making transductions that are not valid in the language — as well as **undergeneration** —

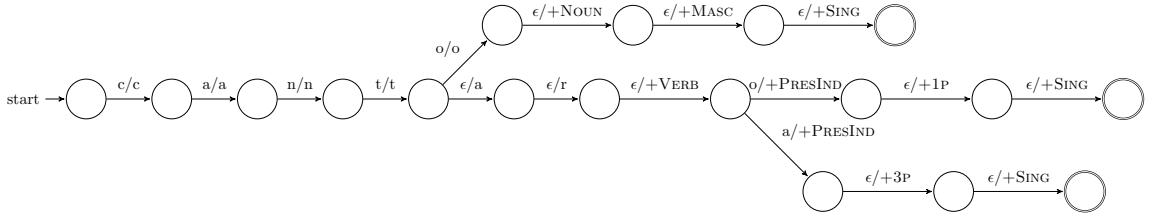


Figure 9.7: Fragment of a finite state transducer for Spanish morphology. There are two accepting paths for the input *canto*: *canto+NOUN+MASC+SING* (masculine singular noun, meaning a song), and *cantar+VERB+PRESIND+1P+SING* (I sing). There is also an accepting path for *canta*, with output *cantar+VERB+PRESIND+3P+SING* (he/she sings).

4590 failing to accept strings or transductions that are valid. For example, a pluralization trans-  
 4591 ducer that does not accept *foot/feet* would undergenerate. Suppose we “fix” the transducer  
 4592 to accept this example, but as a side effect, it now accepts *boot/beet*; the transducer would  
 4593 then be said to overgenerate. A transducer that accepts *foot/foots* but not *foot/feet* would  
 4594 both overgenerate and undergenerate.

#### 4595 9.1.4.4 Finite state composition

4596 Designing finite state transducers to capture the full range of morphological phenomena  
 4597 in any real language is a huge task. Modularization is a classic computer science approach  
 4598 for this situation: decompose a large and unwieldy problem into a set of subproblems,  
 4599 each of which will hopefully have a concise solution. Finite state automata can be mod-  
 4600 ularized through **composition**: feeding the output of one transducer  $T_1$  as the input to  
 4601 another transducer  $T_2$ , written  $T_2 \circ T_1$ . Formally, if there exists some  $y$  such that  $(x, y) \in T_1$   
 4602 (meaning that  $T_1$  produces output  $y$  on input  $x$ ), and  $(y, z) \in T_2$ , then  $(x, z) \in (T_2 \circ T_1)$ .  
 4603 Because finite state transducers are closed under composition, there is guaranteed to be  
 4604 a single finite state transducer that  $T_3 = T_2 \circ T_1$ , which can be constructed as a machine  
 4605 with one state for each pair of states in  $T_1$  and  $T_2$  (Mohri et al., 2002).

4606 **Example: Morphology and orthography** In English morphology, the suffix *-ed* is added  
 4607 to signal the past tense for many verbs: *cook*→*cooked*, *want*→*wanted*, etc. However, English  
 4608 **orthography** dictates that this process cannot produce a spelling with consecutive e’s, so  
 4609 that *bake*→*baked*, not *bakeed*. A modular solution is to build separate transducers for mor-  
 4610 phology and orthography. The morphological transducer  $T_M$  transduces from *bake+PAST*  
 4611 to *bake+ed*, with the + symbol indicating a segment boundary. The input alphabet of  $T_M$   
 4612 includes the lexicon of words and the set of morphological features; the output alphabet  
 4613 includes the characters *a-z* and the + boundary marker. Next, an orthographic transducer  
 4614  $T_O$  is responsible for the transductions *cook+ed*→*cooked*, and *bake+ed*→*baked*. The input  
 4615 alphabet of  $T_O$  must be the same as the output alphabet for  $T_M$ , and the output alphabet

4616 is simply the characters *a-z*. The composed transducer ( $T_O \circ T_M$ ) then transduces from  
 4617 *bake*+PAST to the spelling *baked*. The design of  $T_O$  is left as an exercise.

**Example: Hidden Markov models** Hidden Markov models (chapter 7) can be viewed as weighted finite state transducers, and they can be constructed by transduction. Recall that a hidden Markov model defines a joint probability over words and tags,  $p(w, y)$ , which can be computed as a path through a **trellis** structure. This trellis is itself a weighted finite state acceptor, with edges between all adjacent nodes  $q_{m-1,i} \rightarrow q_{m,j}$  on input  $Y_m = j$ . The edge weights are log-probabilities,

$$\delta(q_{m-1,i}, Y_m = j, q_{m,j}) = \log p(w_m, Y_m = j \mid Y_{m-1} = i) \quad [9.20]$$

$$= \log p(w_m \mid Y_m = j) + \log \Pr(Y_m = j \mid Y_{m-1} = i). \quad [9.21]$$

4618 Because there is only one possible transition for each tag  $Y_m$ , this WFSA is deterministic.  
 4619 The score for any tag sequence  $\{y_m\}_{m=1}^M$  is the sum of these log-probabilities, correspond-  
 4620 ing to the total log probability  $\log p(w, y)$ . Furthermore, the trellis can be constructed by  
 4621 the composition of simpler FSTs.

- 4622 • First, construct a “transition” transducer to represent a bigram probability model  
 4623 over tag sequences,  $T_T$ . This transducer is almost identical to the  $n$ -gram language  
 4624 model acceptor in § 9.1.3.1: there is one state for each tag, and the edge weights  
 4625 equal to the transition log-probabilities,  $\delta(q_i, j, j, q_j) = \log \Pr(Y_m = j \mid Y_{m-1} = i)$ .  
 4626 Note that  $T_T$  is a transducer, with identical input and output at each arc; this makes  
 4627 it possible to compose  $T_T$  with other transducers.
- 4628 • Next, construct an “emission” transducer to represent the probability of words given  
 4629 tags,  $T_E$ . This transducer has only a single state, with arcs for each word/tag pair,  
 4630  $\delta(q_0, i, j, q_0) = \log \Pr(W_m = j \mid Y_m = i)$ . The input vocabulary is the set of all tags,  
 4631 and the output vocabulary is the set of all words.
- 4632 • The composition  $T_E \circ T_T$  is a finite state transducer with one state per tag, as shown  
 4633 in Figure 9.8. Each state has  $V \times K$  outgoing edges, representing transitions to each  
 4634 of the  $K$  other states, with outputs for each of the  $V$  words in the vocabulary. The  
 4635 weights for these edges are equal to,

$$\delta(q_i, Y_m = j, w_m, q_j) = \log p(w_m, Y_m = j \mid Y_{m-1} = i). \quad [9.22]$$

- 4636 • The trellis is a structure with  $M \times K$  nodes, for each of the  $M$  words to be tagged and  
 4637 each of the  $K$  tags in the tagset. It can be built by composition of  $(T_E \circ T_T)$  against an  
 4638 unweighted **chain FSA**  $M_A(w)$  that is specially constructed to accept only a given  
 4639 input  $w_1, w_2, \dots, w_M$ , shown in Figure 9.9. The trellis for input  $w$  is built from the  
 4640 composition  $M_A(w) \circ (T_E \circ T_T)$ . Composing with the unweighted  $M_A(w)$  does not  
 4641 affect the edge weights from  $(T_E \circ T_T)$ , but it selects the subset of paths that generate  
 4642 the word sequence  $w$ .

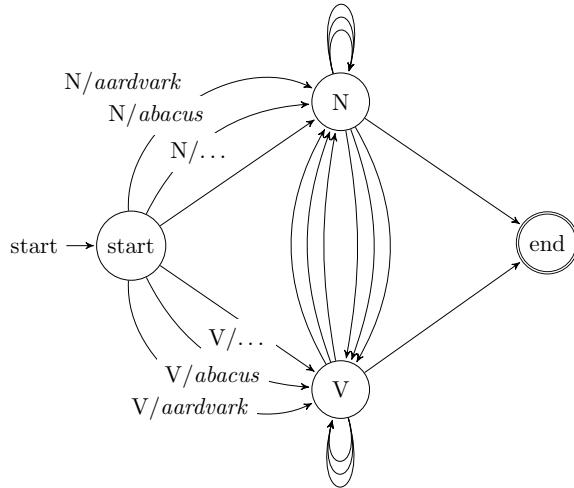


Figure 9.8: Finite state transducer for hidden Markov models, with a small tagset of nouns and verbs. For each pair of tags (including self-loops), there is an edge for every word in the vocabulary. For simplicity, input and output are only shown for the edges from the start state. Weights are also omitted from the diagram; for each edge from  $q_i$  to  $q_j$ , the weight is equal to  $\log p(w_m, Y_m = j \mid Y_{m-1} = i)$ , except for edges to the end state, which are equal to  $\log \Pr(Y_m = \diamond \mid Y_{m-1} = i)$ .

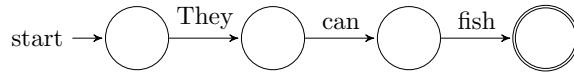


Figure 9.9: Chain finite state acceptor for the input *They can fish*.

#### 4643 9.1.5 \*Learning weighted finite state automata

4644 In generative models such as  $n$ -gram language models and hidden Markov models, the  
 4645 edge weights correspond to log probabilities, which can be obtained from relative fre-  
 4646 quency estimation. However, in other cases, we wish to learn the edge weights from in-  
 4647 put/output pairs. This is difficult in non-deterministic finite state automata, because we  
 4648 do not observe the specific arcs that are traversed in accepting the input, or in transducing  
 4649 from input to output. The path through the automaton is a **latent variable**.

4650 Chapter 5 presented one method for learning with latent variables: expectation max-  
 4651 imization (EM). This involves computing a distribution  $q(\cdot)$  over the latent variable, and  
 4652 iterating between updates to this distribution and updates to the parameters — in this  
 4653 case, the arc weights. The **forward-backward algorithm** (§ 7.5.3.3) describes a dynamic  
 4654 program for computing a distribution over arcs in the trellis structure of a hidden Markov

model, but this is a special case of the more general problem for finite state automata. Eisner (2002) describes an **expectation semiring**, which enables the expected number of transitions across each arc to be computed through a semiring shortest-path algorithm. Alternative approaches for generative models include Markov Chain Monte Carlo (Chiang et al., 2010) and spectral learning (Balle et al., 2011).

Further afield, we can take a perceptron-style approach, with each arc corresponding to a feature. The classic perceptron update would update the weights by subtracting the difference between the feature vector corresponding to the predicted path and the feature vector corresponding to the correct path. Since the path is not observed, we resort to a **hidden variable perceptron**. The model is described formally in § 12.4, but the basic idea is to compute an update from the difference between the features from the predicted path and the features for the best-scoring path that generates the correct output.

## 9.2 Context-free languages

Beyond the class of regular languages lie the context-free languages. An example of a language that is context-free but not finite state is the set of arithmetic expressions with balanced parentheses. Intuitively, to accept only strings in this language, an FSA would have to “count” the number of left parentheses, and make sure that they are balanced against the number of right parentheses. An arithmetic expression can be arbitrarily long, yet by definition an FSA has a finite number of states. Thus, for any FSA, there will be a string that with too many parentheses to count. More formally, the **pumping lemma** is a proof technique for showing that languages are not regular. It is typically demonstrated for the simpler case  $a^n b^n$ , the language of strings containing a sequence of  $a$ 's, and then an equal-length sequence of  $b$ 's.<sup>4</sup>

There are at least two arguments for the relevance of non-regular formal languages to linguistics. First, there are natural language phenomena that are argued to be isomorphic to  $a^n b^n$ . For English, the classic example is **center embedding**, shown in Figure 9.10. The initial expression *the dog* specifies a single dog. Embedding this expression into *the cat ... chased* specifies a particular cat — the one chased by the dog. This cat can then be embedded again to specify a goat, in the less felicitous but arguably grammatical expression, *the goat the cat the dog chased kissed*, which refers to the goat who was kissed by the cat which was chased by the dog. Chomsky (1957) argues that to be grammatical, a center-embedded construction must be balanced: if it contains  $n$  noun phrases (e.g., *the cat*), they must be followed by exactly  $n - 1$  verbs. An FSA that could recognize such expressions would also be capable of recognizing the language  $a^n b^n$ . Because we can prove that no FSA exists for  $a^n b^n$ , no FSA can exist for center embedded constructions either. En-

---

<sup>4</sup>Details 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

4690 glish includes center embedding, and so the argument goes, English grammar as a whole  
 4691 cannot be regular.<sup>5</sup>

4692 A more practical argument for moving beyond regular languages is modularity. Many  
 4693 linguistic phenomena — especially in syntax — involve constraints that apply at long  
 4694 distance. Consider the problem of determiner-noun number agreement in English: we  
 4695 can say *the coffee* and *these coffees*, but not *\*these coffee*. By itself, this is easy enough to model  
 4696 in an FSA. However, fairly complex modifying expressions can be inserted between the  
 4697 determiner and the noun:

- 4698 (9.5) the burnt coffee
- 4699 (9.6) the badly-ground coffee
- 4700 (9.7) the burnt and badly-ground Italian coffee
- 4701 (9.8) these burnt and badly-ground Italian coffees
- 4702 (9.9) \*these burnt and badly-ground Italian coffee

4703 Again, an FSA can be designed to accept modifying expressions such as *burnt and badly-*  
 4704 *ground Italian*. Let's call this FSA  $F_M$ . To reject the final example, a finite state acceptor  
 4705 must somehow "remember" that the determiner was plural when it reaches the noun *cof-*  
 4706 *fee* at the end of the expression. The only way to do this is to make two identical copies  
 4707 of  $F_M$ : one for singular determiners, and one for plurals. While this is possible in the  
 4708 finite state framework, it is inconvenient — especially in languages where more than one  
 4709 attribute of the noun is marked by the determiner. **Context-free languages** facilitate mod-  
 4710 ularity across such long-range dependencies.

### 4711 9.2.1 Context-free grammars

4712 Context-free languages are specified by **context-free grammars (CFGs)**, which are tuples  
 4713  $(N, \Sigma, R, S)$  consisting of:

---

<sup>5</sup>The 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

- 4714 • a finite set of **non-terminals**  $N$ ;
- 4715 • a finite alphabet  $\Sigma$  of **terminal symbols**;
- 4716 • a set of **production rules**  $R$ , each of the form  $A \rightarrow \beta$ , where  $A \in N$  and  $\beta \in (\Sigma \cup N)^*$ ;
- 4717 • a designated start symbol  $S$ .

4718 In the production rule  $A \rightarrow \beta$ , the left-hand side (LHS)  $A$  must be a non-terminal;  
 4719 the right-hand side (RHS) can be a sequence of terminals or non-terminals,  $\{n, \sigma\}^*, n \in$   
 4720  $N, \sigma \in \Sigma$ . A non-terminal can appear on the left-hand side of many production rules.  
 4721 A non-terminal can appear on both the left-hand side and the right-hand side; this is a  
 4722 **recursive production**, and is analogous to self-loops in finite state automata. The name  
 4723 “context-free” is based on the property that the production rule depends only on the LHS,  
 4724 and not on its ancestors or neighbors; this is analogous to Markov property of finite state  
 4725 automata, in which the behavior at each step depends only on the current state, on not on  
 4726 the path by which that state was reached.

4727 A **derivation**  $\tau$  is a sequence of steps from the start symbol  $S$  to a surface string  $w \in \Sigma^*$ ,  
 4728 which is the **yield** of the derivation. A string  $w$  is in a context-free language if there is  
 4729 some derivation from  $S$  yielding  $w$ . **Parsing** is the problem of finding a derivation for a  
 4730 string in a grammar. Algorithms for parsing are described in chapter 10.

4731 Like regular expressions, context-free grammars define the language but not the com-  
 4732 putation necessary to recognize it. The context-free analogues to finite state acceptors are  
 4733 **pushdown automata**, a theoretical model of computation in which input symbols can be  
 4734 pushed onto a stack with potentially infinite depth. For more details, see Sipser (2012).

### 4735 9.2.1.1 Example

4736 Figure 9.11 shows a context-free grammar for arithmetic expressions such as  $1 + 2 \div 3 - 4$ .  
 4737 In this grammar, the terminal symbols include the digits  $\{1, 2, \dots, 9\}$  and the op-  
 4738 erators  $\{+, -, \times, \div\}$ . The rules include the  $|$  symbol, a notational convenience that makes  
 4739 it possible to specify multiple right-hand sides on a single line: the statement  $A \rightarrow x | y$

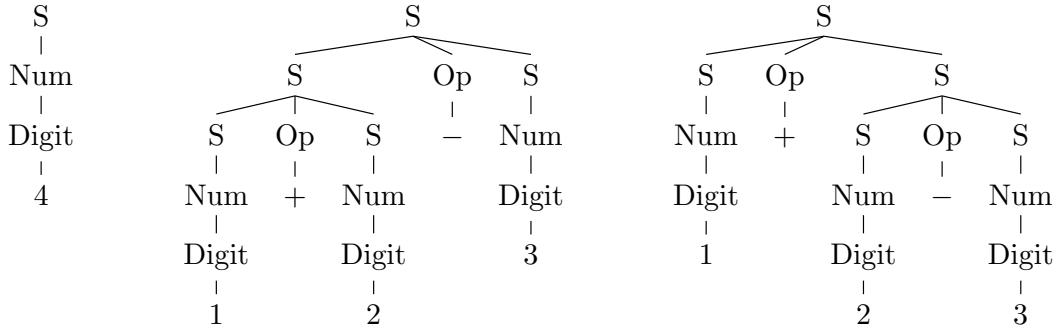


Figure 9.12: Some example derivations from the arithmetic grammar in Figure 9.11

4740 defines *two* productions,  $A \rightarrow x$  and  $A \rightarrow y$ . This grammar is recursive: the non-termals  $S$   
4741 and  $\text{NUM}$  can produce themselves.

4742 Derivations are typically shown as trees, with production rules applied from the top  
4743 to the bottom. The tree on the left in Figure 9.12 describes the derivation of a single digit,  
4744 through the sequence of productions  $S \rightarrow \text{NUM} \rightarrow \text{DIGIT} \rightarrow 4$  (these are all **unary produc-**  
4745 **tions**, because the right-hand side contains a single element). The other two trees in  
4746 Figure 9.12 show alternative derivations of the string  $1 + 2 - 3$ . The existence of multiple  
4747 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.<sup>6</sup> For the two derivations of  $1 + 2 - 3$  in Figure 9.12, the notation is:

$$(S (S (S (\text{Num} (Digit 1))) (\text{Op} +) (S (\text{Num} (Digit 2))))) (\text{Op} -) (S (\text{Num} (Digit 3)))) \quad [9.23]$$

$$(S (S (\text{Num} (Digit 1))) (\text{Op} +) (S (\text{Num} (Digit 2)) (\text{Op} -) (S (\text{Num} (Digit 3)))))). \quad [9.24]$$

#### 4748 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  $n > 0$ .

$$\begin{aligned} S &\rightarrow aSb \mid ab \\ S &\rightarrow aSb \mid aabb \mid ab \end{aligned}$$

4749 Two grammars are **weakly equivalent** if they generate the same strings. Two grammars  
4750 are **strongly equivalent** if they generate the same strings via the same derivations. The  
4751 grammars above are only weakly equivalent.

<sup>6</sup>This 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:

$$A \rightarrow BC$$

$$A \rightarrow a$$

- 4752 All CFGs can be converted into a CNF grammar that is weakly equivalent. To convert a  
 4753 grammar into CNF, we first address productions that have more than two non-terminals  
 4754 on the RHS by creating new “dummy” non-terminals. For example, if we have the pro-  
 4755 duction,

$$W \rightarrow X Y Z, \quad [9.25]$$

it is replaced with two productions,

$$W \rightarrow X W \setminus X \quad [9.26]$$

$$W \setminus X \rightarrow Y Z. \quad [9.27]$$

- 4756 In these productions,  $W \setminus X$  is a new dummy non-terminal. This transformation **binarizes**  
 4757 the grammar, which is critical for efficient bottom-up parsing, as we will see in chapter 10.  
 4758 Productions whose right-hand side contains a mix of terminal and non-terminal symbols  
 4759 can be replaced in a similar fashion.

- 4760 Unary non-terminal productions  $A \rightarrow B$  are replaced as follows: identify all produc-  
 4761 tions  $B \rightarrow \alpha$ , and add  $A \rightarrow \alpha$  to the grammar. For example, in the grammar described in  
 4762 Figure 9.11, we would replace  $\text{NUM} \rightarrow \text{DIGIT}$  with  $\text{NUM} \rightarrow 1 \mid 2 \mid \dots \mid 9$ . However, we  
 4763 keep the production  $\text{NUM} \rightarrow \text{NUM DIGIT}$ , which is a valid binary production.

### 4764 9.2.2 Natural language syntax as a context-free language

- 4765 Context-free grammars are widely used to represent **syntax**, which is the set of rules that  
 4766 determine whether an utterance is judged to be grammatical. If this representation were  
 4767 perfectly faithful, then a natural language such as English could be transformed into a  
 4768 formal language, consisting of exactly the (infinite) set of strings that would be judged to  
 4769 be grammatical by a fluent English speaker. We could then build parsing software that  
 4770 would automatically determine if a given utterance were grammatical.<sup>7</sup>

- 4771 Contemporary theories generally do *not* consider natural languages to be context-free  
 4772 (see § 9.3), yet context-free grammars are widely used in natural language parsing. The  
 4773 reason is that context-free representations strike a good balance: they cover a broad range  
 4774 of syntactic phenomena, and they can be parsed efficiently. This section therefore de-  
 4775 scribes how to handle a core fragment of English syntax in context-free form, following

---

<sup>7</sup>You are encouraged to move beyond this cursory treatment of syntax by consulting a textbook on linguistics (e.g., Akmajian et al., 2010; Bender, 2013).

4776 the conventions of the **Penn Treebank** (PTB; Marcus et al., 1993), a large-scale annotation  
 4777 of English language syntax. The generalization to “mildly” context-sensitive languages is  
 4778 discussed in § 9.3.

4779 The Penn Treebank annotation is a **phrase-structure grammar** of English. This means  
 4780 that sentences are broken down into **constituents**, which are contiguous sequences of  
 4781 words that function as coherent units for the purpose of linguistic analysis. Constituents  
 4782 generally have a few key properties:

4783 **Movement.** Constituents can often be moved around sentences as units.

- 4784 (9.10) Abigail gave (her brother) (a fish).  
 4785 (9.11) Abigail gave (a fish) to (her brother).

4786 In contrast, *gave her* and *brother a* cannot easily be moved while preserving gram-  
 4787 maticality.

4788 **Substitution.** Constituents can be substituted by other phrases of the same type.

- 4789 (9.12) Max thanked (his older sister).  
 4790 (9.13) Max thanked (her).

4791 In contrast, substitution is not possible for other contiguous units like *Max thanked*  
 4792 and *thanked his*.

4793 **Coordination.** Coordinators like *and* and *or* can conjoin constituents.

- 4794 (9.14) (Abigail) and (her younger brother) bought a fish.  
 4795 (9.15) Abigail (bought a fish) and (gave it to Max).  
 4796 (9.16) Abigail (bought) and (greedily ate) a fish.

4797 Units like *brother bought* and *bought a* cannot easily be coordinated.

4798 These examples argue for units such as *her brother* and *bought a fish* to be treated as con-  
 4799 stituents. Other sequences of words in these examples, such as *Abigail gave* and *brother*  
 4800 *a fish*, cannot be moved, substituted, and coordinated in these ways. In phrase-structure  
 4801 grammar, constituents are nested, so that *the senator from New Jersey* contains the con-  
 4802 stituent *from New Jersey*, which in turn contains *New Jersey*. The sentence itself is the max-  
 4803 imal constituent; each word is a minimal constituent, derived from a unary production  
 4804 from a part-of-speech tag. Between part-of-speech tags and sentences are **phrases**. In  
 4805 phrase-structure grammar, phrases have a type that is usually determined by their **head**  
 4806 **word**: for example, a **noun phrase** corresponds to a noun and the group of words that

4807 modify it, such as *her younger brother*; a **verb phrase** includes the verb and its modifiers,  
4808 such as *bought a fish* and *greedily ate it*.

4809 In context-free grammars, each phrase type is a non-terminal, and each constituent is  
4810 the substring that the non-terminal yields. Grammar design involves choosing the right  
4811 set of non-terminals. Fine-grained non-terminals make it possible to represent more fine-  
4812 grained linguistic phenomena. For example, by distinguishing singular and plural noun  
4813 phrases, it is possible to have a grammar of English that generates only sentences that  
4814 obey subject-verb agreement. However, enforcing subject-verb agreement is considerably  
4815 more complicated in languages like Spanish, where the verb must agree in both person  
4816 and number with subject. In general, grammar designers must trade off between **over-**  
4817 **generation** — a grammar that permits ungrammatical sentences — and **undergeneration**  
4818 — a grammar that fails to generate grammatical sentences. Furthermore, if the grammar is  
4819 to support manual annotation of syntactic structure, it must be simple enough to annotate  
4820 efficiently.

### 4821 9.2.3 A phrase-structure grammar for English

4822 To better understand how phrase-structure grammar works, let's consider the specific  
4823 case of the Penn Treebank grammar of English. The main phrase categories in the Penn  
4824 Treebank (PTB) are based on the main part-of-speech classes: noun phrase (NP), verb  
4825 phrase (VP), prepositional phrase (PP), adjectival phrase (ADJP), and adverbial phrase  
4826 (ADVP). The top-level category is S, which conveniently stands in for both “sentence”  
4827 and the “start” symbol. **Complement clauses** (e.g., *I take the good old fashioned ground that*  
4828 *the whale is a fish*) are represented by the non-terminal SBAR. The terminal symbols in  
4829 the grammar are individual words, which are generated from unary productions from  
4830 part-of-speech tags (the PTB tagset is described in § 8.1).

4831 This section explores the productions from the major phrase-level categories, explaining  
4832 how to generate individual tag sequences. The production rules are approached in a  
4833 “theory-driven” manner: first the syntactic properties of each phrase type are described,  
4834 and then some of the necessary production rules are listed. But it is important to keep  
4835 in mind that the Penn Treebank was produced in a “data-driven” manner. After the set  
4836 of non-terminals was specified, annotators were free to analyze each sentence in what-  
4837 ever way seemed most linguistically accurate, subject to some high-level guidelines. The  
4838 grammar of the Penn Treebank is simply the set of productions that were required to ana-  
4839 lyze the several million words of the corpus. By design, the grammar overgenerates — it  
4840 does not exclude ungrammatical sentences.

4841 **9.2.3.1 Sentences**

The most common production rule for sentences is,

$$S \rightarrow NP VP \quad [9.28]$$

which accounts for simple sentences like *Abigail ate the kimchi* — as we will see, the direct object *the kimchi* is part of the verb phrase. But there are more complex forms of sentences as well:

$$S \rightarrow ADVP NP VP \quad \begin{matrix} Unfortunately Abigail ate the kimchi. \\ Abigail ate the kimchi and Max had a burger. \end{matrix} \quad [9.29]$$

$$S \rightarrow S CC S \quad \begin{matrix} Abigail ate the kimchi and Max had a burger. \\ Eat the kimchi. \end{matrix} \quad [9.30]$$

$$S \rightarrow VP \quad \begin{matrix} Eat the kimchi. \end{matrix} \quad [9.31]$$

- 4842 where ADVP is an adverbial phrase (e.g., *unfortunately*, *very unfortunately*) and CC is a  
 4843 coordinating conjunction (e.g., *and*, *but*).<sup>8</sup>

4844 **9.2.3.2 Noun phrases**

Noun phrases refer to entities, real or imaginary, physical or abstract: *Asha*, *the steamed dumpling*, *parts and labor*, *nobody*, *the whiteness of the whale*, and *the rise of revolutionary syndicalism in the early twentieth century*. Noun phrase productions include “bare” nouns, which may optionally follow determiners, as well as pronouns:

$$NP \rightarrow NN | NNS | NNP | PRP \quad [9.32]$$

$$NP \rightarrow DET NN | DET NNS | DET NNP \quad [9.33]$$

- 4845 The tags NN, NNS, and NNP refer to singular, plural, and proper nouns; PRP refers to  
 4846 personal pronouns, and DET refers to determiners. The grammar also contains terminal  
 4847 productions from each of these tags, e.g.,  $PRP \rightarrow I | you | we | \dots$ .

Noun phrases may be modified by adjectival phrases (ADJP; e.g., *the small Russian dog*) and numbers (CD; e.g., *the five pastries*), each of which may optionally follow a determiner:

$$NP \rightarrow ADJP NN | ADJP NNS | DET ADJP NN | DET ADJP NNS \quad [9.34]$$

$$NP \rightarrow CD NNS | DET CD NNS | \dots \quad [9.35]$$

Some noun phrases include multiple nouns, such as *the liberation movement* and *an antelope horn*, necessitating additional productions:

$$NP \rightarrow NN NN | NN NNS | DET NN NN | \dots \quad [9.36]$$

---

<sup>8</sup>Notice 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.

4848 These multiple noun constructions can be combined with adjectival phrases and cardinal  
 4849 numbers, leading to a large number of additional productions.

Recursive noun phrase productions include coordination, prepositional phrase attachment, subordinate clauses, and verb phrase adjuncts:

$NP \rightarrow NP\ CC\ NP$	<i>e.g., the red and the black</i>	[9.37]
$NP \rightarrow NP\ PP$	<i>e.g., the President of the Georgia Institute of Technology</i>	[9.38]
$NP \rightarrow NP\ SBAR$	<i>e.g., a whale which he had wounded</i>	[9.39]
$NP \rightarrow NP\ VP$	<i>e.g., a whale taken near Shetland</i>	[9.40]

4850 These recursive productions are a major source of ambiguity, because the VP and PP non-  
 4851 terminals can also generate NP children. Thus, the *the President of the Georgia Institute of*  
 4852 *Technology* can be derived in two ways, as can *a whale taken near Shetland in October*.

4853 But aside from these few recursive productions, the noun phrase fragment of the Penn  
 4854 Treebank grammar is relatively flat, containing a large of number of productions that go  
 4855 from NP directly to a sequence of parts-of-speech. If noun phrases had more internal  
 4856 structure, the grammar would need fewer rules, which, as we will see, would make pars-  
 4857 ing faster and machine learning easier. Vadas and Curran (2011) propose to add additional  
 4858 structure in the form of a new non-terminal called a **nominal modifier** (NML), e.g.,

4859 (9.17) (NP (NN crude) (NN oil) (NNS prices)) (PTB analysis)  
 4860 (NP (NML (NN crude) (NN oil)) (NNS prices)) (NML-style analysis)

4861 Another proposal is to treat the determiner as the head of a **determiner phrase** (DP;  
 4862 Abney, 1987). There are linguistic arguments for and against determiner phrases (e.g.,  
 4863 Van Eynde, 2006). From the perspective of context-free grammar, DPs enable more struc-  
 4864 tured analyses of some constituents, e.g.,

4865 (9.18) (NP (DT the) (JJ white) (NN whale)) (PTB analysis)  
 4866 (DP (DT the) (NP (JJ white) (NN whale))) (DP-style analysis).

### 4867 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*).<sup>9</sup> Each of these forms can constitute a verb phrase on its

---

<sup>9</sup>This tagset is specific to English: for example, VBP is a meaningful category only because English morphology distinguishes third-person singular from all person-number combinations.

own:

$$VP \rightarrow VB \mid VBZ \mid VBD \mid VBN \mid VBG \mid VBP \quad [9.41]$$

More complex verb phrases can be formed by a number of recursive productions, including the use of coordination, modal verbs (MD; *she should snack*), and the infinitival *to* (TO):

$VP \rightarrow MD \ VP$	<i>She will snack</i>	[9.42]
$VP \rightarrow VBD \ VP$	<i>She had snacked</i>	[9.43]
$VP \rightarrow VBZ \ VP$	<i>She has been snacking</i>	[9.44]
$VP \rightarrow VBN \ VP$	<i>She has been snacking</i>	[9.45]
$VP \rightarrow TO \ VP$	<i>She wants to snack</i>	[9.46]
$VP \rightarrow VP \ CC \ VP$	<i>She buys and eats many snacks</i>	[9.47]

- 4868 Each of these productions uses recursion, with the VP non-terminal appearing in both the  
 4869 LHS and RHS. This enables the creation of complex verb phrases, such as *She will have*  
 4870 *wanted to have been snacking*.

Transitive verbs take noun phrases as direct objects, and ditransitive verbs take two direct objects:

$VP \rightarrow VBZ \ NP$	<i>She teaches algebra</i>	[9.48]
$VP \rightarrow VBG \ NP$	<i>She has been teaching algebra</i>	[9.49]
$VP \rightarrow VBD \ NP \ NP$	<i>She taught her brother algebra</i>	[9.50]

These productions are *not* recursive, so a unique production is required for each verb part-of-speech. They also do not distinguish transitive from intransitive verbs, so the resulting grammar overgenerates examples like *\*She sleeps sushi* and *\*She learns Boyang algebra*. Sentences can also be direct objects:

$VP \rightarrow VBZ \ S$	<i>Asha wants to eat the kimchi</i>	[9.51]
$VP \rightarrow VBZ \ SBAR$	<i>Asha knows that Boyang eats the kimchi</i>	[9.52]

- 4871 The first production overgenerates, licensing sentences like *\*Asha sees Boyang eats the kimchi*.  
 4872 This problem could be addressed by designing a more specific set of sentence non-  
 4873 terminals, indicating whether the main verb can be conjugated.

Verbs can also be modified by prepositional phrases and adverbial phrases:

$VP \rightarrow VBZ \ PP$	<i>She studies at night</i>	[9.53]
$VP \rightarrow VBZ \ ADVP$	<i>She studies intensively</i>	[9.54]
$VP \rightarrow ADVP \ VBG$	<i>She is not studying</i>	[9.55]

4874 Again, because these productions are not recursive, the grammar must include productions  
 4875 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]

4876 The PTB does not have a special non-terminal for copular verbs, so this production generates  
 4877 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]

4878 As the second production shows, particle productions are required for all configurations  
 4879 of verb parts-of-speech and direct objects.

#### 4880 9.2.3.4 Other constituents

The remaining constituents require far fewer productions. **Prepositional phrases** almost always consist of a preposition and a noun phrase,

$PP \rightarrow IN\ NP$  *the whiteness of the whale* [9.60]

$PP \rightarrow TO\ NP$  *What the white whale was to Ahab, has been hinted.* [9.61]

Similarly, complement clauses consist of a complementizer (usually a preposition, possibly null) and a sentence,

$SBAR \rightarrow IN\ S$  *She said that it was spicy* [9.62]

$SBAR \rightarrow S$  *She said it was spicy* [9.63]

Adverbial phrases are usually bare adverbs ( $ADVP \rightarrow RB$ ), with a few exceptions:

$ADVP \rightarrow RB\ RBR$  *They went considerably further* [9.64]

$ADVP \rightarrow ADVP\ PP$  *They went considerably further than before* [9.65]

4881 The tag RBR is a comparative adverb.

Adjectival phrases extend beyond bare adjectives ( $\text{ADJP} \rightarrow \text{JJ}$ ) in a number of ways:

$\text{ADJP} \rightarrow \text{RB JJ}$	<i>very hungry</i>	[9.66]
$\text{ADJP} \rightarrow \text{RBR JJ}$	<i>more hungry</i>	[9.67]
$\text{ADJP} \rightarrow \text{JJS JJ}$	<i>best possible</i>	[9.68]
$\text{ADJP} \rightarrow \text{RB JJR}$	<i>even bigger</i>	[9.69]
$\text{ADJP} \rightarrow \text{JJ CC JJ}$	<i>high and mighty</i>	[9.70]
$\text{ADJP} \rightarrow \text{JJ JJ}$	<i>West German</i>	[9.71]
$\text{ADJP} \rightarrow \text{RB VBN}$	<i>previously reported</i>	[9.72]

4882 The tags JJR and JJS refer to comparative and superlative adjectives respectively.

All of these phrase types can be coordinated:

$\text{PP} \rightarrow \text{PP CC PP}$	<i>on time and under budget</i>	[9.73]
$\text{ADVP} \rightarrow \text{ADVP CC ADVP}$	<i>now and two years ago</i>	[9.74]
$\text{ADJP} \rightarrow \text{ADJP CC ADJP}$	<i>quaint and rather deceptive</i>	[9.75]
$\text{SBar} \rightarrow \text{SBar CC SBar}$	<i>whether they want control</i>	[9.76]
	<i>or whether they want exports</i>	

#### 4883 9.2.4 Grammatical ambiguity

4884 Context-free parsing is useful not only because it determines whether a sentence is gram-  
 4885 matical, but mainly because the constituents and their relations can be applied to tasks  
 4886 such as information extraction (chapter 17) and sentence compression (Jing, 2000; Clarke  
 4887 and Lapata, 2008). However, the **ambiguity** of wide-coverage natural language grammars  
 4888 poses a serious problem for such potential applications. As an example, Figure 9.13 shows  
 4889 two possible analyses for the simple sentence *We eat sushi with chopsticks*, depending on  
 4890 whether the *chopsticks* modify *eat* or *sushi*. Realistic grammars can license thousands or  
 4891 even millions of parses for individual sentences. **Weighted context-free grammars** solve  
 4892 this problem by attaching weights to each production, and selecting the derivation with  
 4893 the highest score. This is the focus of chapter 10.

#### 4894 9.3 \*Mildly context-sensitive languages

4895 Beyond context-free languages lie **context-sensitive languages**, in which the expansion  
 4896 of a non-terminal depends on its neighbors. In the general class of context-sensitive  
 4897 languages, computation becomes much more challenging: the membership problem for  
 4898 context-sensitive languages is PSPACE-complete. Since PSPACE contains the complexity  
 4899 class NP (problems that can be solved in polynomial time on a non-deterministic Turing

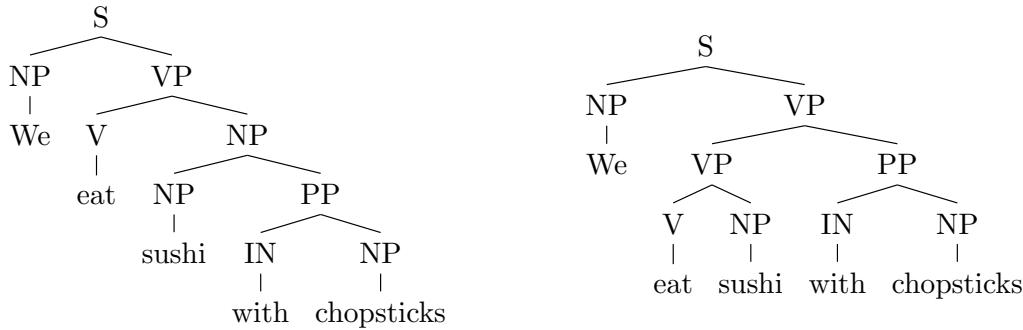


Figure 9.13: Two derivations of the same sentence

4900 machine), PSPACE-complete problems cannot be solved efficiently if  $P \neq NP$ . Thus, de-  
 4901 signing an efficient parsing algorithm for the full class of context-sensitive languages is  
 4902 probably hopeless.<sup>10</sup>

4903 However, Joshi (1985) identifies a set of properties that define **mildly context-sensitive**  
 4904 **languages**, which are a strict subset of context-sensitive languages. Like context-free lan-  
 4905 guages, mildly context-sensitive languages are efficiently parseable. However, the mildly  
 4906 context-sensitive languages include non-context-free languages, such as the “copy lan-  
 4907 guage”  $\{ww \mid w \in \Sigma^*\}$  and the language  $a^m b^n c^m d^n$ . Both are characterized by **cross-**  
 4908 **serial dependencies**, linking symbols at long distance across the string.<sup>11</sup> For example, in  
 4909 the language  $a^n b^m c^n d^m$ , each  $a$  symbol is linked to exactly one  $c$  symbol, regardless of the  
 4910 number of intervening  $b$  symbols.

### 4911 9.3.1 Context-sensitive phenomena in natural language

4912 Such phenomena are occasionally relevant to natural language. A classic example is found  
 4913 in Swiss-German (Shieber, 1985), in which sentences such as *we let the children help Hans*  
 4914 *paint the house* are realized by listing all nouns before all verbs, i.e., *we the children Hans the*  
 4915 *house let help paint*. Furthermore, each noun’s determiner is dictated by the noun’s **case**  
 4916 **marking** (the role it plays with respect to the verb). Using an argument that is analogous  
 4917 to the earlier discussion of center-embedding (§ 9.2), Shieber argues that these case mark-  
 4918 ing constraints are a cross-serial dependency, homomorphic to  $a^m b^n c^m d^n$ , and therefore  
 4919 not context-free.

<sup>10</sup>If  $P \neq NP$ , then it contains problems that cannot be solved in polynomial time on a non-deterministic Turing machine; equivalently, solutions to these problems cannot even be checked in polynomial time (Arora and Barak, 2009).

<sup>11</sup>A further condition of the set of mildly-context-sensitive languages is *constant growth*: if the strings in the language are arranged by length, the gap in length between any pair of adjacent strings is bounded by some language specific constant. This condition excludes languages such as  $\{a^{2^n} \mid n \geq 0\}$ .

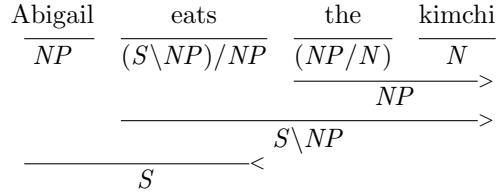


Figure 9.14: A syntactic analysis in CCG involving forward and backward function application

As with the move from regular to context-free languages, mildly context-sensitive languages can be motivated by expedience. While infinite sequences of cross-serial dependencies cannot be handled by context-free grammars, even finite sequences of cross-serial dependencies are more convenient to handle using a mildly context-sensitive formalism like **tree-adjoining grammar** (TAG) and **combinatory categorial grammar** (CCG). Furthermore, TAG-inspired parsers have been shown to be particularly effective in parsing the Penn Treebank (Collins, 1997; Carreras et al., 2008), and CCG plays a leading role in current research on semantic parsing (Zettlemoyer and Collins, 2005). Furthermore, these two formalisms are weakly equivalent: any language that can be specified in TAG can also be specified in CCG, and vice versa (Joshi et al., 1991). The remainder of the chapter gives a brief overview of CCG, but you are encouraged to consult Joshi and Schabes (1997) and Steedman and Baldridge (2011) for more detail on TAG and CCG respectively.

### 9.3.2 Combinatory categorial grammar

In combinatory categorial grammar, structural analyses are built up through a small set of generic combinatorial operations, which apply to immediately adjacent sub-structures. These operations act on the categories of the sub-structures, producing a new structure with a new category. The basic categories include S (sentence), NP (noun phrase), VP (verb phrase) and N (noun). The goal is to label the entire span of text as a sentence, S.

Complex categories, or types, are constructed from the basic categories, parentheses, and forward and backward slashes: for example,  $S/NP$  is a complex type, indicating a sentence that is lacking a noun phrase to its right;  $S\backslash NP$  is a sentence lacking a noun phrase to its left. Complex types act as functions, and the most basic combinatory operations are function application to either the right or left neighbor. For example, the type of a verb phrase, such as *eats*, would be  $S\backslash NP$ . Applying this function to a subject noun phrase to its left results in an analysis of *Abigail eats* as category S, indicating a successful parse.

Transitive verbs must first be applied to the direct object, which in English appears to the right of the verb, before the subject, which appears on the left. They therefore have the more complex type  $(S\backslash NP)/NP$ . Similarly, the application of a determiner to the noun at

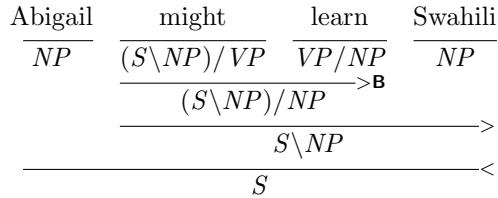


Figure 9.15: A syntactic analysis in CCG involving function composition (example modified from Steedman and Baldridge, 2011)

4949 its right results in a noun phrase, so determiners have the type NP/N. Figure 9.14 pro-  
 4950 vides an example involving a transitive verb and a determiner. A key point from this  
 4951 example is that it can be trivially transformed into phrase-structure tree, by treating each  
 4952 function application as a constituent phrase. Indeed, when CCG's only combinatory op-  
 4953 erators are forward and backward function application, it is equivalent to context-free  
 4954 grammar. However, the location of the "effort" has changed. Rather than designing good  
 4955 productions, the grammar designer must focus on the **lexicon** — choosing the right cate-  
 4956 gories for each word. This makes it possible to parse a wide range of sentences using only  
 4957 a few generic combinatory operators.

4958 Things become more interesting with the introduction of two additional operators:  
 4959 **composition** and **type-raising**. Function composition enables the combination of com-  
 4960 plex types:  $X/Y \circ Y/Z \Rightarrow_B X/Z$  (forward composition) and  $Y\backslash Z \circ X\backslash Y \Rightarrow_B X\backslash Z$  (back-  
 4961 ward composition).<sup>12</sup> Composition makes it possible to "look inside" complex types, and  
 4962 combine two adjacent units if the "input" for one is the "output" for the other. Figure 9.15  
 4963 shows how function composition can be used to handle modal verbs. While this sen-  
 4964 tence can be parsed using only function application, the composition-based analysis is  
 4965 preferable because the unit *might learn* functions just like a transitive verb, as in the exam-  
 4966 ple *Abigail studies Swahili*. This in turn makes it possible to analyze conjunctions such as  
 4967 *Abigail studies and might learn Swahili*, attaching the direct object *Swahili* to the entire con-  
 4968 joined verb phrase *studies and might learn*. The Penn Treebank grammar fragment from  
 4969 § 9.2.3 would be unable to handle this case correctly: the direct object *Swahili* could attach  
 4970 only to the second verb *learn*.

4971 Type raising converts an element of type  $X$  to a more complex type:  $X \Rightarrow_T T/(T\backslash X)$   
 4972 (forward type-raising to type  $T$ ), and  $X \Rightarrow_T T\backslash(T/X)$  (backward type-raising to type  
 4973  $T$ ). Type-raising makes it possible to reverse the relationship between a function and its  
 4974 argument — by transforming the argument into a function over functions over arguments!  
 4975 An example may help. Figure 9.15 shows how to analyze an object relative clause, *a story*  
 4976 *that Abigail tells*. The problem is that *tells* is a transitive verb, expecting a direct object to  
 4977 its right. As a result, *Abigail tells* is not a valid constituent. The issue is resolved by raising

<sup>12</sup>The subscript **B** follows notation from Curry and Feys (1958).

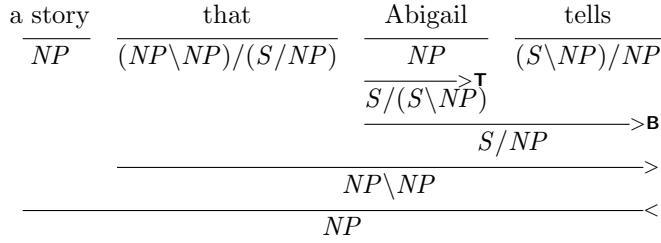


Figure 9.16: A syntactic analysis in CCG involving an object relative clause (based on slides from Alex Clark)

4978 *Abigail* from NP to the complex type  $(S / NP) \setminus NP$ . This function can then be combined  
 4979 with the transitive verb *tells* by forward composition, resulting in the type  $(S / NP)$ , which  
 4980 is a sentence lacking a direct object to its right.<sup>13</sup> From here, we need only design the  
 4981 lexical entry for the complementizer *that* to expect a right neighbor of type  $(S / NP)$ , and  
 4982 the remainder of the derivation can proceed by function application.

4983 Composition and type-raising give CCG considerable power and flexibility, but at a  
 4984 price. The simple sentence *Abigail tells Max* can be parsed in two different ways: by func-  
 4985 tion application (first forming the verb phrase *tells Max*), and by type-raising and compo-  
 4986 sition (first forming the non-constituent *Abigail tells*). This **derivational ambiguity** does  
 4987 not affect the resulting linguistic analysis, so it is sometimes known as **spurious ambi-**  
 4988 **guity**. Hockenmaier and Steedman (2007) present a translation algorithm for converting  
 4989 the Penn Treebank into CCG derivations, using composition and type-raising only when  
 4990 necessary.

## 4991 Exercises

- 4992 1. Sketch out the state diagram for finite-state acceptors for the following languages  
 4993 on the alphabet  $\{a, b\}$ .
- 4994 a) Even-length strings. (Be sure to include 0 as an even number.)
- 4995 b) Strings that contain *aaa* as a substring.
- 4996 c) Strings containing an even number of *a* and an odd number of *b* symbols.
- 4997 d) Strings in which the substring *bbb* must be terminal if it appears — the string  
 4998 need not contain *bbb*, but if it does, nothing can come after it.
- 4999 2. Levenshtein edit distance is the number of insertions, substitutions, or deletions  
 5000 required to convert one string to another.

---

<sup>13</sup>The missing direct object would be analyzed as a **trace** in CFG-like approaches to syntax, including the Penn Treebank.

- 5001        a) Define a finite-state acceptor that accepts all strings with edit distance 1 from  
 5002            the target string, *target*.  
 5003        b) Now think about how to generalize your design to accept all strings with edit  
 5004            distance from the target string equal to  $d$ . If the target string has length  $\ell$ , what  
 5005            is the minimal number of states required?
- 5006     3. Construct an FSA in the style of Figure 9.3, which handles the following examples:

- 5007        • *nation*/N, *national*/ADJ, *nationalize*/V, *nationalizer*/N  
 5008        • *America*/N, *American*/ADJ, *Americanize*/V, *Americanizer*/N

5009     Be sure that your FSA does not accept any further derivations, such as *\*nationalizeral*  
 5010    and *\*Americanizern*.

- 5011     4. Show how to construct a trigram language model in a weighted finite-state acceptor.  
 5012     Make sure that you handle the edge cases at the beginning and end of the sequence  
 5013     accurately.
- 5014     5. Extend the FST in Figure 9.6 to handle the other two parts of rule 1a of the Porter  
 5015     stemmer: *-sses* → *ss*, and *-ies* → *-i*.

- 5016     6. § 9.1.4.4 describes  $T_O$ , a transducer that captures English orthography by transduc-  
 5017        ing *cook + ed* → *cooked* and *bake + ed* → *baked*. Design an unweighted finite-state  
 5018        transducer that captures this property of English orthography.

5019     Next, augment the transducer to appropriately model the suffix *-s* when applied to  
 5020     words ending in *s*, e.g. *kiss+s* → *kisses*.

- 5021     7. Add parenthesization to the grammar in Figure 9.11 so that it is no longer ambigu-  
 5022        ous.
- 5023     8. Construct three examples — a noun phrase, a verb phrase, and a sentence — which  
 5024        can be derived from the Penn Treebank grammar fragment in § 9.2.3, yet are not  
 5025        grammatical. Avoid reusing examples from the text. Optionally, propose corrections  
 5026        to the grammar to avoid generating these cases.
- 5027     9. Produce parses for the following sentences, using the Penn Treebank grammar frag-  
 5028        ment from § 9.2.3.

- 5029        (9.19) This aggression will not stand.  
 5030        (9.20) I can get you a toe.  
 5031        (9.21) Sometimes you eat the bar and sometimes the bar eats you.

5032     Then produce parses for three short sentences from a news article from this week.

5033 10. \* One advantage of CCG is its flexibility in handling coordination:

5034 (9.22) *Abigail and Max speak Swahili*

5035 (9.23) *Abigail speaks and Max understands Swahili*

Define the lexical entry for *and* as

$$\textit{and} := (X/X) \setminus X, \quad [9.77]$$

5036 where  $X$  can refer to any type. Using this lexical entry, show how to parse the two  
5037 examples above. In the second example, *Swahili* should be combined with the coor-  
5038 dination *Abigail speaks and Max understands*, and not just with the verb *understands*.

5039 

# Chapter 10

5040 

## Context-free parsing

5041 Parsing is the task of determining whether a string can be derived from a given context-  
5042 free grammar, and if so, how. The parse structure can answer basic questions of who-did-  
5043 what-to-whom, and is useful for various downstream tasks, such as semantic analysis  
5044 (chapter 12 and 13) and information extraction (chapter 17).

For a given input and grammar, how many parse trees are there? Consider a minimal context-free grammar with only one non-terminal,  $X$ , and the following productions:

$$\begin{aligned} X &\rightarrow X \ X \\ X &\rightarrow aardvark \mid abacus \mid \dots \mid zyther \end{aligned}$$

The second line indicates unary productions to every nonterminal in  $\Sigma$ . In this grammar, the number of possible derivations for a string  $w$  is equal to the number of binary bracketings, e.g.,

$$(((w_1 w_2) w_3) w_4) w_5), \quad (((w_1 (w_2 w_3)) w_4) w_5), \quad ((w_1 (w_2 (w_3 w_4))) w_5), \quad \dots$$

5045 The number of such bracketings is a **Catalan number**, which grows super-exponentially  
5046 in the length of the sentence,  $C_n = \frac{(2n)!}{(n+1)n!}$ . As with sequence labeling, it is only possible to  
5047 exhaustively search the space of parses by resorting to locality assumptions, which make it  
5048 possible to search efficiently by reusing shared substructures with dynamic programming.  
5049 This chapter focuses on a bottom-up dynamic programming algorithm, which enables  
5050 exhaustive search of the space of possible parses, but imposes strict limitations on the  
5051 form of scoring function. These limitations can be relaxed by abandoning exhaustive  
5052 search. Non-exact search methods will be briefly discussed at the end of this chapter, and  
5053 one of them — **transition-based parsing** — will be the focus of chapter 11.

S	$\rightarrow$	NP VP
NP	$\rightarrow$	NP PP   <i>we</i>   <i>sushi</i>   <i>chopsticks</i>
PP	$\rightarrow$	IN NP
IN	$\rightarrow$	<i>with</i>
VP	$\rightarrow$	V NP   VP PP
V	$\rightarrow$	<i>eat</i>

Table 10.1: A toy example context-free grammar

5054 **10.1 Deterministic bottom-up parsing**

5055 The **CKY algorithm**<sup>1</sup> is a bottom-up approach to parsing in a context-free grammar. It  
 5056 efficiently tests whether a string is in a language, without enumerating all possible parses.  
 5057 The algorithm first forms small constituents, and then tries to merge them into larger  
 5058 constituents.

5059 To understand the algorithm, consider the input, *We eat sushi with chopsticks*. According-  
 5060 ing to the toy grammar in Table 10.1, each terminal symbol can be generated by exactly  
 5061 one unary production, resulting in the sequence NP V NP IN NP. Next, we try to apply  
 5062 binary productions to merge adjacent symbols into larger constituents: for example, V  
 5063 NP can be merged into a verb phrase (VP), and IN NP can be merged into a prepositional  
 5064 phrase (PP). Bottom-up parsing tries to find some series of mergers that ultimately results  
 5065 in the start symbol S covering the entire input.

5066 The CKY algorithm systematizes this approach, incrementally constructing a table  $t$  in  
 5067 which each cell  $t[i, j]$  contains the set of nonterminals that can derive the span  $w_{i+1:j}$ . The  
 5068 algorithm fills in the upper right triangle of the table; it begins with the diagonal, which  
 5069 corresponds to substrings of length 1, and then computes derivations for progressively  
 5070 larger substrings, until reaching the upper right corner  $t[0, M]$ , which corresponds to the  
 5071 entire input,  $w_{1:M}$ . If the start symbol S is in  $t[0, M]$ , then the string  $w$  is in the language  
 5072 defined by the grammar. This process is detailed in Algorithm 13, and the resulting data  
 5073 structure is shown in Figure 10.1. Informally, here's how it works:

- 5074 • Begin by filling in the diagonal: the cells  $t[m - 1, m]$  for all  $m \in \{1, 2, \dots, M\}$ . These  
 5075 cells are filled with terminal productions that yield the individual tokens; for the  
 5076 word  $w_2 = \text{sushi}$ , we fill in  $t[1, 2] = \{\text{NP}\}$ , and so on.
- 5077 • Then fill in the next diagonal, in which each cell corresponds to a subsequence of  
 5078 length two:  $t[0, 2], t[1, 3], \dots, t[M - 2, M]$ . These cells are filled in by looking for  
 5079 binary productions capable of producing at least one entry in each of the cells corre-

---

<sup>1</sup>The name is for Cocke-Kasami-Younger, the inventors of the algorithm. It is a special case **chart parsing**, because its stores reusable computations in a chart-like data structure.

---

**Algorithm 13** The CKY algorithm for parsing a sequence  $w \in \Sigma^*$  in a context-free grammar  $G = (N, \Sigma, R, S)$ , with non-terminals  $N$ , production rules  $R$ , and start symbol  $S$ . The grammar is assumed to be in Chomsky normal form (§ 9.2.1.2). The function  $\text{PICKFROM}(b[i, j, X])$  selects an element of the set  $b[i, j, X]$  arbitrarily. All values of  $t$  and  $b$  are initialized to  $\emptyset$ .

---

```

1: procedure CKY( $w, G = (N, \Sigma, R, S)$ )
2:   for  $m \in \{1 \dots M\}$  do
3:      $t[m - 1, m] \leftarrow \{X : (X \rightarrow w_m) \in R\}$ 
4:   for  $\ell \in \{2, 3, \dots, M\}$  do                                 $\triangleright$  Iterate over constituent lengths
5:     for  $m \in \{0, 1, \dots, M - \ell\}$  do           $\triangleright$  Iterate over left endpoints
6:       for  $k \in \{m + 1, m + 2, \dots, m + \ell - 1\}$  do       $\triangleright$  Iterate over split points
7:         for  $(X \rightarrow Y Z) \in R$  do           $\triangleright$  Iterate over rules
8:           if  $Y \in t[m, k] \wedge Z \in t[k, m + \ell]$  then
9:              $t[m, m + \ell] \leftarrow t[m, m + \ell] \cup X$            $\triangleright$  Add non-terminal to table
10:             $b[m, m + \ell, X] \leftarrow b[m, m + \ell, X] \cup (Y, Z, k)$        $\triangleright$  Add back-pointers
11:   if  $S \in t[0, M]$  then
12:     return TRACEBACK( $S, 0, M, b$ )
13:   else
14:     return  $\emptyset$ 
15: procedure TRACEBACK( $X, i, j, b$ )
16:   if  $j = i + 1$  then
17:     return  $X$ 
18:   else
19:      $(Y, Z, k) \leftarrow \text{PICKFROM}(b[i, j, X])$ 
20:     return  $X \rightarrow (\text{TRACEBACK}(Y, i, k, b), \text{TRACEBACK}(Z, k, j, b))$ 

```

---

5080 sponding to left and right children. For example, the cell  $t[1, 3]$  includes VP because  
 5081 the grammar includes the production  $\text{VP} \rightarrow \text{V NP}$ , and the chart contains  $\text{V} \in t[1, 2]$   
 5082 and  $\text{NP} \in t[2, 3]$ .

- 5083 • At the next diagonal, the entries correspond to spans of length three. At this level,  
 5084 there is an additional decision at each cell: where to split the left and right children.  
 5085 The cell  $t[i, j]$  corresponds to the subsequence  $w_{i+1:j}$ , and we must choose some  
 5086 *split point*  $i < k < j$ , so that  $w_{i+1:k}$  is the left child and  $w_{k+1:j}$  is the right child. We  
 5087 consider all possible  $k$ , looking for productions that generate elements in  $t[i, k]$  and  
 5088  $t[k, j]$ ; the left-hand side of all such productions can be added to  $t[i, j]$ . When it is  
 5089 time to compute  $t[i, j]$ , the cells  $t[i, k]$  and  $t[k, j]$  are guaranteed to be complete, since  
 5090 these cells correspond to shorter sub-strings of the input.

- 5091 • The process continues until we reach  $t[0, M]$ .

5092 Figure 10.1 shows the chart that arises from parsing the sentence *We eat sushi with chop-*  
 5093 *sticks* using the grammar defined above.

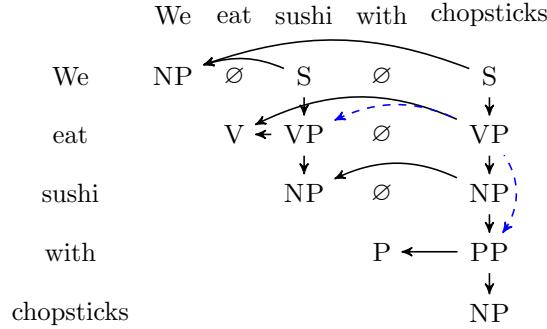


Figure 10.1: An example completed CKY chart. The solid and dashed lines show the back pointers resulting from the two different derivations of VP in position  $t[1, 5]$ .

### 5094 10.1.1 Recovering the parse tree

5095 As with the Viterbi algorithm, it is possible to identify a successful parse by storing and  
 5096 traversing an additional table of back-pointers. If we add an entry  $X$  to cell  $t[i, j]$  by using  
 5097 the production  $X \rightarrow YZ$  and the split point  $k$ , then we store the back-pointer  $b[i, j, X] =$   
 5098  $(Y, Z, k)$ . Once the table is complete, we can recover a parse by tracing this pointers,  
 5099 starting at  $b[0, M, S]$ , and stopping when they ground out at terminal productions.

5100 For ambiguous sentences, there will be multiple paths to reach  $S \in t[0, M]$ . For exam-  
 5101 ple, in Figure 10.1, the goal state  $S \in t[0, M]$  is reached through the state  $VP \in t[1, 5]$ , and  
 5102 there are two different ways to generate this constituent: one with *(eat sushi)* and *(with  
 5103 chopsticks)* as children, and another with *(eat)* and *(sushi with chopsticks)* as children. The  
 5104 presence of multiple paths indicates that the input can be generated by the grammar in  
 5105 more than one way. In Algorithm 13, one of these derivations is selected arbitrarily. As  
 5106 discussed in § 10.3, **weighted context-free grammars** can select a single parse that maxi-  
 5107 mizes a scoring function.

### 5108 10.1.2 Non-binary productions

5109 The CKY algorithm assumes that all productions with non-terminals on the right-hand  
 5110 side (RHS) are binary. But in real grammars, such as the one considered in chapter 9,  
 5111 there will be productions with more than two elements on the right-hand side, and other  
 5112 productions with only a single element.

- 5113 • Productions with more than two elements on the right-hand side can be **binarized**  
 5114 by creating additional non-terminals, as described in § 9.2.1.2. For example, given  
 5115 the production  $VP \rightarrow V NP NP$  (for ditransitive verbs), we can convert to  $VP \rightarrow$   
 5116  $VP_{ditrans}/NP NP$ , and then add the production  $VP_{ditrans}/NP \rightarrow V NP$ .

- What about unary productions like  $VP \rightarrow V$ ? In practice, this is handled by making a second pass on each diagonal, in which each cell  $t[i, j]$  is augmented with all possible unary productions capable of generating each item already in the cell — formally,  $t[i, j]$  is extended to its **unary closure**. Suppose the example grammar in Table 10.1 were extended to include the production  $VP \rightarrow V$ , enabling sentences with intransitive verb phrases, like *we eat*. Then the cell  $t[1, 2]$  — corresponding to the word *eat* — would first include the set  $\{V\}$ , and would be augmented to the set  $\{V, VP\}$  during this second pass.

### 10.1.3 Complexity

For an input of length  $M$  and a grammar with  $R$  productions and  $N$  non-terminals, the space complexity of the CKY algorithm is  $\mathcal{O}(M^2N)$ : the number of cells in the chart is  $\mathcal{O}(M^2)$ , and each cell must hold  $\mathcal{O}(N)$  elements. The time complexity is  $\mathcal{O}(M^3R)$ : each cell is computed by searching over  $\mathcal{O}(M)$  split points, with  $R$  possible productions for each split point. Both the time and space complexity are considerably worse than the Viterbi algorithm, which is linear in the length of the input.

## 10.2 Ambiguity

Syntactic ambiguity is endemic to natural language. Here are a few broad categories:

- **Attachment ambiguity:** e.g., *We eat sushi with chopsticks, I shot an elephant in my pajamas*. In these examples, the prepositions (*with, in*) can attach to either the verb or the direct object.
- **Modifier scope:** e.g., *southern food store, plastic cup holder*. In these examples, the first word could be modifying the subsequent adjective, or the final noun.
- **Particle versus preposition:** e.g., *The puppy tore up the staircase*. Phrasal verbs like *tore up* often include particles which could also act as prepositions. This has structural implications: if *up* is a preposition, then *up the staircase* is a prepositional phrase; if *up* is a particle, then *the staircase* is the direct object to the verb.
- **Complement structure:** e.g., *The students complained to the professor that they didn't understand*. This is another form of attachment ambiguity, where the complement *that they didn't understand* could attach to the main verb (*complained*), or to the indirect object (*the professor*).
- **Coordination scope:** e.g., *"I see," said the blind man, as he picked up the hammer and saw*. In this example, the lexical ambiguity for *saw* enables it to be coordinated either with the noun *hammer* or the verb *picked up*.

These forms of ambiguity can combine, so that seemingly simple headlines like *Fed raises interest rates* have dozens of possible analyses even in a minimal grammar. In a broad coverage grammar, typical sentences can have millions of parses. While careful grammar design can chip away at this ambiguity, a better strategy is to combine broad coverage parsers with data driven strategies for identifying the correct analysis.

### 10.2.1 Parser evaluation

Before continuing to parsing algorithms that are able to handle ambiguity, we stop to consider how to measure parsing performance. Suppose we have a set of *reference parses* — the ground truth — and a set of *system parses* that we would like to score. A simple solution would be per-sentence accuracy: the parser is scored by the proportion of sentences on which the system and reference parses exactly match.<sup>2</sup> 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.

---

<sup>2</sup>Most 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  $\leq 40$ , and on 62% of parses of length  $\leq 15$  in the Penn Treebank.

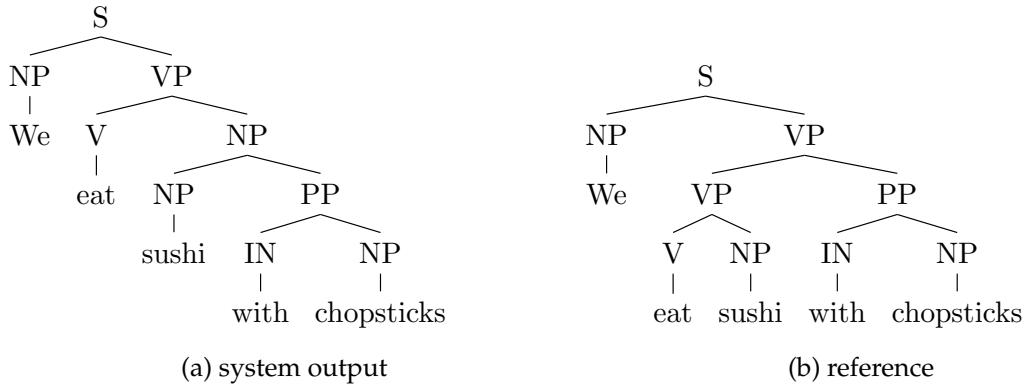


Figure 10.2: Two possible analyses from the grammar in Table 10.1

5179 The labeled and unlabeled precision of this parse is  $\frac{3}{4} = 0.75$ , and the recall is  $\frac{3}{4} = 0.75$ , for  
 5180 an F-measure of 0.75. For an example in which precision and recall are not equal, suppose  
 5181 the reference parse instead included the production  $VP \rightarrow V NP PP$ . In this parse, the  
 5182 reference does not contain the constituent  $w_{2:3}$ , so the recall would be 1.<sup>3</sup>

## 5183 10.2.2 Local solutions

5184 Some ambiguity can be resolved locally. Consider the following examples,

5185 (10.1) We met the President on Monday.

5186 (10.2) We met the President of Mexico.

Each case ends with a preposition, which can be attached to the verb *met* or the noun phrase *the president*. This ambiguity can be resolved by using a labeled corpus to compare the likelihood of observing the preposition alongside each candidate attachment point,

$$p(on \mid met) \geq p(on \mid President) \quad [10.1]$$

$$p(of \mid met) \geq p(of \mid President). \quad [10.2]$$

5187 A comparison of these probabilities would successfully resolve this case (Hindle and  
5188 Rooth, 1993). Other cases, such as the example ... *eat sushi with chopsticks*, require consider-  
5189 ing the object of the preposition — consider the alternative ... *eat sushi with soy sauce*. With  
5190 sufficient labeled data, the problem of prepositional phrase attachment can be treated as  
5191 a classification task (Ratnaparkhi et al., 1994).

---

<sup>3</sup>While 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.

5192     However, there are inherent limitations to local solutions. While toy examples may  
 5193     have just a few ambiguities to resolve, realistic sentences have thousands or millions of  
 5194     possible parses. Furthermore, attachment decisions are interdependent, as shown in the  
 5195     garden path example:

5196     (10.3) Cats scratch people with claws with knives.

5197     We may want to attach *with claws* to *scratch*, as would be correct in the shorter sentence  
 5198     in *cats scratch people with claws*. But this leaves nowhere to attach *with knives*. The cor-  
 5199     rect interpretation can be identified only by considering the attachment decisions jointly.  
 5200     The huge number of potential parses may seem to make exhaustive search impossible.  
 5201     But as with sequence labeling, locality assumptions make it possible to search this space  
 5202     efficiently.

### 5203 10.3 Weighted Context-Free Grammars

5204     Let us define a derivation  $\tau$  as a set of **anchored productions**,

$$\tau = \{X \rightarrow \alpha, (i, j, k)\}, \quad [10.3]$$

5205     with  $X$  corresponding to the left-hand side non-terminal and  $\alpha$  corresponding to the right-  
 5206     hand side. For grammars in Chomsky normal form,  $\alpha$  is either a pair of non-terminals or  
 5207     a terminal symbol. The indices  $i, j, k$  anchor the production in the input, with  $X$  deriving  
 5208     the span  $w_{i+1:j}$ . For binary productions,  $w_{i+1:k}$  indicates the span of the left child, and  
 5209      $w_{k+1:j}$  indicates the span of the right child; for unary productions,  $k$  is ignored. For an  
 5210     input  $w$ , the optimal parse is then,

$$\hat{\tau} = \underset{\tau \in \mathcal{T}(w)}{\operatorname{argmax}} \Psi(\tau), \quad [10.4]$$

5211     where  $\mathcal{T}(w)$  is the set of derivations that yield the input  $w$ .

5212     The scoring function  $\Psi$  decomposes across anchored productions,

$$\Psi(\tau) = \sum_{(X \rightarrow \alpha, (i, j, k)) \in \tau} \psi(X \rightarrow \alpha, (i, j, k)). \quad [10.5]$$

5213     This is a locality assumption, akin to the assumption in Viterbi sequence labeling. In this  
 5214     case, the assumption states that the overall score is a sum over scores of productions,  
 5215     which are computed independently. In a **weighted context-free grammar** (WCFG), the  
 5216     score of each anchored production  $X \rightarrow (\alpha, i, j, k)$  is simply  $\psi(X \rightarrow \alpha)$ , ignoring the  
 5217     anchors  $(i, j, k)$ . In other parsing models, the anchors can be used to access features of the  
 5218     input, while still permitting efficient bottom-up parsing.

		$\psi(\cdot)$	$\exp \psi(\cdot)$
S	$\rightarrow \text{NP VP}$	0	1
NP	$\rightarrow \text{NP PP}$	-1	$\frac{1}{2}$
	$\rightarrow \text{we}$	-2	$\frac{1}{4}$
	$\rightarrow \text{sushi}$	-3	$\frac{1}{8}$
	$\rightarrow \text{chopsticks}$	-3	$\frac{1}{8}$
PP	$\rightarrow \text{IN NP}$	0	1
IN	$\rightarrow \text{with}$	0	1
VP	$\rightarrow \text{V NP}$	-1	$\frac{1}{2}$
	$\rightarrow \text{VP PP}$	-2	$\frac{1}{4}$
	$\rightarrow \text{MD V}$	-2	$\frac{1}{4}$
V	$\rightarrow \text{eat}$	0	1

Table 10.2: An example weighted context-free grammar (WCFG). The weights are chosen so that  $\exp \psi(\cdot)$  sums to one over right-hand sides for each non-terminal; this is required by probabilistic context-free grammars, but not by WCFGs in general.

**Example** Consider the weighted grammar shown in Table 10.2, and the analysis in Figure 10.2b.

$$\begin{aligned} \Psi(\tau) &= \psi(S \rightarrow \text{NP VP}) + \psi(VP \rightarrow \text{VP PP}) + \psi(VP \rightarrow \text{V NP}) + \psi(PP \rightarrow \text{IN NP}) \\ &\quad + \psi(NP \rightarrow \text{We}) + \psi(V \rightarrow \text{eat}) + \psi(NP \rightarrow \text{sushi}) + \psi(IN \rightarrow \text{with}) + \psi(NP \rightarrow \text{chopsticks}) \end{aligned} \quad [10.6]$$

$$= 0 - 2 - 1 + 0 - 2 + 0 - 3 + 0 - 3 = -11. \quad [10.7]$$

5219 In the alternative parse in Figure 10.2a, the production  $VP \rightarrow VP PP$  (with score -2) is  
 5220 replaced with the production  $NP \rightarrow NP PP$  (with score -1); all other productions are the  
 5221 same. As a result, the score for this parse is -10.

5222 This example hints at a big problem with WCFG parsing on non-terminals such as  
 5223 NP, VP, and PP: a WCFG will *always* prefer either VP or NP attachment, without regard  
 5224 to what is being attached! This problem is addressed in § 10.5.

### 5225 10.3.1 Parsing with weighted context-free grammars

5226 The optimization problem in Equation 10.4 can be solved by modifying the CKY algo-  
 5227 rithm. In the deterministic CKY algorithm, each cell  $t[i, j]$  stored a set of non-terminals  
 5228 capable of deriving the span  $w_{i+1:j}$ . We now augment the table so that the cell  $t[i, j, X]$   
 5229 is the *score of the best derivation of  $w_{i+1:j}$  from non-terminal  $X$* . This score is computed  
 5230 recursively: for the anchored binary production  $(X \rightarrow Y Z, (i, j, k))$ , we compute:

---

**Algorithm 14** CKY algorithm for parsing a string  $w \in \Sigma^*$  in a weighted context-free grammar  $(N, \Sigma, R, S)$ , where  $N$  is the set of non-terminals and  $R$  is the set of weighted productions. The grammar is assumed to be in Chomsky normal form (§ 9.2.1.2). The function TRACEBACK is defined in Algorithm 13.

---

```

procedure WCKY( $w, G = (N, \Sigma, R, S)$ )
  for all  $i, j, X$  do ▷ Initialization
     $t[i, j, X] \leftarrow 0$ 
     $b[i, j, X] \leftarrow \emptyset$ 
  for  $m \in \{1, 2, \dots, M\}$  do
    for all  $X \in N$  do
       $t[m, m + 1, X] \leftarrow \psi(X \rightarrow w_m, (m, m + 1, m))$ 
  for  $\ell \in \{2, 3, \dots, M\}$  do
    for  $m \in \{0, 1, \dots, M - \ell\}$  do
      for  $k \in \{m + 1, m + 2, \dots, m + \ell - 1\}$  do
         $t[m, m + \ell, X] \leftarrow \max_{k, Y, Z} \psi(X \rightarrow Y Z, (m, m + \ell, k)) + t[m, k, Y] + t[k, m + \ell, Z]$ 
         $b[m, m + \ell, X] \leftarrow \operatorname{argmax}_{k, Y, Z} \psi(X \rightarrow Y Z, (m + \ell, k)) + t[m, k, Y] + t[k, m + \ell, Z]$ 
  return TRACEBACK( $S, 0, M, b$ )

```

---

- 5231     • the score of the anchored production,  $\psi(X \rightarrow Y Z, (i, j, k))$ ;
- 5232     • the score of the best derivation of the left child,  $t[i, k, Y]$ ;
- 5233     • the score of the best derivation of the right child,  $t[k, j, Z]$ .

5234 These scores are combined by addition. As in the unscored CKY algorithm, the table  
 5235 is constructed by considering spans of increasing length, so the scores for spans  $t[i, k, Y]$   
 5236 and  $t[k, j, Z]$  are guaranteed to be available at the time we compute the score  $t[i, j, X]$ . The  
 5237 value  $t[0, M, S]$  is the score of the best derivation of  $w$  from the grammar. Algorithm 14  
 5238 formalizes this procedure.

5239 As in unweighted CKY, the parse is recovered from the table of back pointers  $b$ , where  
 5240 each  $b[i, j, X]$  stores the argmax split point  $k$  and production  $X \rightarrow Y Z$  in the derivation of  
 5241  $w_{i+1:j}$  from  $X$ . The best parse can be obtained by tracing these pointers backwards from  
 5242  $b[0, M, S]$ , all the way to the terminal symbols. This is analogous to the computation of the  
 5243 best sequence of labels in the Viterbi algorithm by tracing pointers backwards from the  
 5244 end of the trellis. Note that we need only store back-pointers for the *best* path to  $t[i, j, X]$ ;  
 5245 this follows from the locality assumption that the global score for a parse is a combination  
 5246 of the local scores of each production in the parse.

**Example** Let's revisit the parsing table in Figure 10.1. In a weighted CFG, each cell would include a score for each non-terminal; non-terminals that cannot be generated are

---

**Algorithm 15** Generative model for derivations from probabilistic context-free grammars in Chomsky Normal Form (CNF).

---

```

procedure DRAWSUBTREE(X)
    sample  $(X \rightarrow \alpha) \sim p(\alpha | X)$ 
    if  $\alpha = (Y Z)$  then
        return DRAWSUBTREE(Y)  $\cup$  DRAWSUBTREE(Z)
    else
        return  $(X \rightarrow \alpha)$             $\triangleright$  In CNF, all unary productions yield terminal symbols

```

---

assumed to have a score of  $-\infty$ . The first diagonal contains the scores of unary productions:  $t[0, 1, \text{NP}] = -2$ ,  $t[1, 2, \text{V}] = 0$ , and so on. At the next diagonal, we compute the scores for spans of length 2:  $t[1, 3, \text{VP}] = -1 + 0 - 3 = -4$ ,  $t[3, 5, \text{PP}] = 0 + 0 - 3 = -3$ , and so on. Things get interesting when we reach the cell  $t[1, 5, \text{VP}]$ , which contains the score for the derivation of the span  $w_{2:5}$  from the non-terminal VP. This score is computed as a max over two alternatives,

$$t[1, 5, \text{VP}] = \max(\psi(\text{VP} \rightarrow \text{VP PP}, (1, 3, 5)) + t[1, 3, \text{VP}] + t[3, 5, \text{PP}], \\ \psi(\text{VP} \rightarrow \text{V NP}, (1, 2, 5)) + t[1, 2, \text{V}] + t[2, 5, \text{NP}]) \quad [10.8]$$

$$= \max(-2 - 4 - 3, -1 + 0 - 7) = -8. \quad [10.9]$$

5247 Since the second case is the argmax, we set the back-pointer  $b[1, 5, \text{VP}] = (\text{V}, \text{NP}, 2)$ , enabling the optimal derivation to be recovered.

5249 **10.3.2 Probabilistic context-free grammars**

5250 **Probabilistic context-free grammars (PCFGs)** are a special case of weighted context-  
5251 free grammars that arises when the weights correspond to probabilities. Specifically, the  
5252 weight  $\psi(X \rightarrow \alpha, (i, j, k)) = \log p(\alpha | X)$ , where the probability of the right-hand side  
5253  $\alpha$  is conditioned on the non-terminal  $X$ . These probabilities must be normalized over all  
5254 possible right-hand sides, so that  $\sum_\alpha p(\alpha | X) = 1$ , for all  $X$ . For a given parse  $\tau$ , the prod-  
5255 uct of the probabilities of the productions is equal to  $p(\tau)$ , under the **generative model**  
5256  $\tau \sim \text{DRAWSUBTREE}(S)$ , where the function DRAWSUBTREE is defined in Algorithm 15.

5257 The conditional probability of a parse given a string is,

$$p(\tau | w) = \frac{p(\tau)}{\sum_{\tau' \in \mathcal{T}(w)} p(\tau')} = \frac{\exp \Psi(\tau)}{\sum_{\tau' \in \mathcal{T}(w)} \exp \Psi(\tau')}, \quad [10.10]$$

5258 where  $\Psi(\tau) = \sum_{X \rightarrow \alpha, (i, j, k) \in \tau} \psi(X \rightarrow \alpha)$ ; the anchor is ignored. Because the probability  
5259 is monotonic in the score  $\Psi(\tau)$ , the maximum likelihood parse can be identified by the  
5260 CKY algorithm without modification. If a normalized probability  $p(\tau | w)$  is required,  
5261 the denominator of Equation 10.10 can be computed by the **inside recurrence**, described  
5262 below.

**Example** The WCFG in Table 10.2 is designed so that the weights are log-probabilities, satisfying the constraint  $\sum_{\alpha} \exp \psi(X \rightarrow \alpha) = 1$ . As noted earlier, there are two parses in  $\mathcal{T}(\text{we eat sushi with chopsticks})$ , with scores  $\Psi(\tau_1) = \log p(\tau_1) = -10$  and  $\Psi(\tau_2) = \log p(\tau_2) = -11$ . Therefore, the conditional probability  $p(\tau_1 | \mathbf{w})$  is equal to,

$$p(\tau_1 | \mathbf{w}) = \frac{p(\tau_1)}{p(\tau_1) + p(\tau_2)} = \frac{\exp \Psi(\tau_1)}{\exp \Psi(\tau_1) + \exp \Psi(\tau_2)} = \frac{2^{-10}}{2^{-10} + 2^{-11}} = \frac{2}{3}. \quad [10.11]$$

5263 **The inside recurrence** The denominator of Equation 10.10 can be viewed as a language  
5264 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:

$$t[i, j, X] = \log \sum_{k, Y, Z} \exp (\psi(X \rightarrow Y Z) + t[i, k, Y] + t[k, j, Z]) \quad [10.13]$$

$$= \log \sum_{k, Y, Z} \exp (\log p(Y Z | X) + \log p(Y \rightarrow \mathbf{w}_{i+1:k}) + \log p(Z \rightarrow \mathbf{w}_{k+1:j})) \quad [10.14]$$

$$= \log \sum_{k, Y, Z} p(Y Z | X) \times p(Y \rightarrow \mathbf{w}_{i+1:k}) \times p(Z \rightarrow \mathbf{w}_{k+1:j}) \quad [10.15]$$

$$= \log \sum_{k, Y, Z} p(Y Z, \mathbf{w}_{i+1:k}, \mathbf{w}_{k+1:j} | X) \quad [10.16]$$

$$= \log p(X \rightarrow \mathbf{w}_{i+1:j}). \quad [10.17]$$

5265 This is called the **inside recurrence**, because it computes the probability of each subtree  
5266 as a combination of the probabilities of the smaller subtrees that are inside of it. The  
5267 name implies a corresponding **outside recurrence**, which computes the probability of  
5268 a non-terminal  $X$  spanning  $\mathbf{w}_{i+1:j}$ , joint with the outside context  $(\mathbf{w}_{1:i}, \mathbf{w}_{j+1:M})$ . This  
5269 recurrence is described in § 10.4.3. The inside and outside recurrences are analogous to the  
5270 forward and backward recurrences in probabilistic sequence labeling (see § 7.5.3.3). They  
5271 can be used to compute the marginal probabilities of individual anchored productions,  
5272  $p(X \rightarrow \alpha, (i, j, k) | \mathbf{w})$ , summing over all possible derivations of  $\mathbf{w}$ .

5273 **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:

$$t[i, j, X] = \bigoplus_{k, Y, Z} \psi(X \rightarrow Y Z, (i, j, k)) \otimes t[i, k, Y] \otimes t[k, j, Z]. \quad [10.18]$$

5274 This recurrence subsumes all of the algorithms that we have encountered in this chapter.

5275 **Unweighted CKY.** When  $\psi(X \rightarrow \alpha, (i, j, k))$  is a *Boolean truth value*  $\{\top, \perp\}$ ,  $\otimes$  is logical  
5276 conjunction, and  $\bigoplus$  is logical disjunction, then we derive the CKY recurrence for  
5277 unweighted context-free grammars, discussed in § 10.1 and Algorithm 13.

5278 **Weighted CKY.** When  $\psi(X \rightarrow \alpha, (i, j, k))$  is a scalar score,  $\otimes$  is addition, and  $\bigoplus$  is maxi-  
5279 maximization, then we derive the CKY recurrence for weighted context-free grammars,  
5280 discussed in § 10.3 and Algorithm 14. When  $\psi(X \rightarrow \alpha, (i, j, k)) = \log p(\alpha \mid X)$ ,  
5281 this same setting derives the CKY recurrence for finding the maximum likelihood  
5282 derivation in a probabilistic context-free grammar.

5283 **Inside recurrence.** When  $\psi(X \rightarrow \alpha, (i, j, k))$  is a log probability,  $\otimes$  is addition, and  $\bigoplus =$   
5284  $\log \sum \exp$ , then we derive the inside recurrence for probabilistic context-free gram-  
5285 mmars, discussed in § 10.3.2. It is also possible to set  $\psi(X \rightarrow \alpha, (i, j, k))$  directly equal  
5286 to the probability  $p(\alpha \mid X)$ . In this case,  $\otimes$  is multiplication, and  $\bigoplus$  is addition.  
5287 While this may seem more intuitive than working with log probabilities, there is the  
5288 risk of underflow on long inputs.

5289 Regardless of how the scores are combined, the key point is the locality assumption:  
5290 the score for a derivation is the combination of the independent scores for each anchored  
5291 production, and these scores do not depend on any other part of the derivation. For exam-  
5292 ple, if two non-terminals are siblings, the scores of productions from these non-terminals  
5293 are computed independently. This locality assumption is analogous to the first-order  
5294 Markov assumption in sequence labeling, where the score for transitions between tags  
5295 depends only on the previous tag and current tag, and not on the history. As with se-  
5296 quence labeling, this assumption makes it possible to find the optimal parse efficiently; its  
5297 linguistic limitations are discussed in § 10.5.

5298 **10.4 Learning weighted context-free grammars**

5299 Like sequence labeling, context-free parsing is a form of structure prediction. As a result,  
5300 WCFGs can be learned using the same set of algorithms: generative probabilistic models,  
5301 structured perceptron, maximum conditional likelihood, and maximum margin learning.

5302 In all cases, learning requires a **treebank**, which is a dataset of sentences labeled with  
 5303 context-free parses. Parsing research was catalyzed by the **Penn Treebank** (Marcus et al.,  
 5304 1993), the first large-scale dataset of this type (see § 9.2.2). Phrase structure treebanks exist  
 5305 for roughly two dozen other languages, with coverage mainly restricted to European and  
 5306 East Asian languages, plus Arabic and Urdu.

5307 **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:

$$\psi(X \rightarrow \alpha) = \log p(X \rightarrow \alpha) \quad [10.19]$$

$$\hat{p}(X \rightarrow \alpha) = \frac{\text{count}(X \rightarrow \alpha)}{\text{count}(X)}. \quad [10.20]$$

5308 For example, the probability of the production  $NP \rightarrow DET\ NN$  is the corpus count of  
 5309 this production, divided by the count of the non-terminal  $NP$ . This estimator applies  
 5310 to terminal productions as well: the probability of  $NN \rightarrow whale$  is the count of how often  
 5311 *whale* appears in the corpus as generated from an  $NN$  tag, divided by the total count of the  
 5312  $NN$  tag. Even with the largest treebanks — currently on the order of one million tokens  
 5313 — it is difficult to accurately compute probabilities of even moderately rare events, such  
 5314 as  $NN \rightarrow whale$ . Therefore, smoothing is critical for making PCFGs effective.

5315 **10.4.2 Feature-based parsing**

5316 The scores for each production can be computed as an inner product of weights and fea-  
 5317 tures,

$$\psi(X \rightarrow \alpha) = \boldsymbol{\theta} \cdot \mathbf{f}(X, \alpha, (i, j, k), \mathbf{w}), \quad [10.21]$$

5318 where the feature vector  $\mathbf{f}(X, \alpha)$  is a function of the left-hand side  $X$ , the right-hand side  
 5319  $\alpha$ , the anchor indices  $(i, j, k)$ , and the input  $\mathbf{w}$ .

5320 The basic feature  $\mathbf{f}(X, \alpha, (i, j, k)) = \{(X, \alpha)\}$  encodes only the identity of the pro-  
 5321 duction itself, which is a discriminatively-trained model with the same expressiveness as  
 5322 a PCFG. Features on anchored productions can include the words that border the span  
 5323  $w_i, w_{j+1}$ , the word at the split point  $w_{k+1}$ , the presence of a verb or noun in the left child  
 5324 span  $w_{i+1:k}$ , and so on (Durrett and Klein, 2015). Scores on anchored productions can be  
 5325 incorporated into CKY parsing without any modification to the algorithm, because it is  
 5326 still possible to compute each element of the table  $t[i, j, X]$  recursively from its immediate  
 5327 children.

5328 Other features can be obtained by grouping elements on either the left-hand or right-  
 5329 hand side: for example it can be particularly beneficial to compute additional features  
 5330 by clustering terminal symbols, with features corresponding to groups of words with  
 5331 similar syntactic properties. The clustering can be obtained from unlabeled datasets that  
 5332 are much larger than any treebank, improving coverage. Such methods are described in  
 5333 chapter 14.

Feature-based parsing models can be estimated using the usual array of discriminative learning techniques. For example, a structure perceptron update can be computed as (Carreras et al., 2008),

$$\mathbf{f}(\tau, \mathbf{w}^{(i)}) = \sum_{(X \rightarrow \alpha, (i, j, k)) \in \tau} \mathbf{f}(X, \alpha, (i, j, k), \mathbf{w}^{(i)}) \quad [10.22]$$

$$\hat{\tau} = \operatorname{argmax}_{\tau \in \mathcal{T}(\mathbf{w})} \mathbf{f}(\tau, \mathbf{w}^{(i)}) \quad [10.23]$$

$$\boldsymbol{\theta} \leftarrow \mathbf{f}(\tau^{(i)}, \mathbf{w}^{(i)}) - \mathbf{f}(\hat{\tau}, \mathbf{w}^{(i)}). \quad [10.24]$$

5334 A margin-based objective can be optimized by selecting  $\hat{\tau}$  through cost-augmented decoding (§ 2.3.2), enforcing a margin of  $\Delta(\hat{\tau}, \tau)$  between the hypothesis and the reference parse,  
 5335 where  $\Delta$  is a non-negative cost function, such as the Hamming loss (Stern et al., 2017). It  
 5336 is also possible to train feature-based parsing models by conditional log-likelihood, as  
 5337 described in the next section.  
 5338

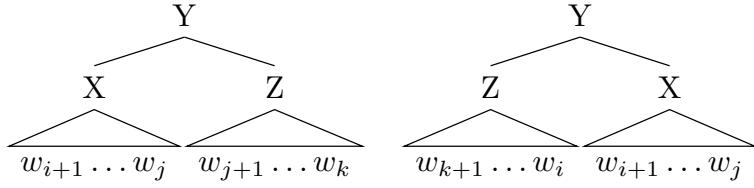
### 5339 10.4.3 \*Conditional random field parsing

5340 The score of a derivation  $\Psi(\tau)$  can be converted into a probability by normalizing over all  
 5341 possible derivations,

$$p(\tau | \mathbf{w}) = \frac{\exp \Psi(\tau)}{\sum_{\tau' \in \mathcal{T}(\mathbf{w})} \exp \Psi(\tau')}. \quad [10.25]$$

5342 Using this probability, a WCFG can be trained by maximizing the conditional log-likelihood  
 5343 of a labeled corpus.

5344 Just as in logistic regression and the conditional random field over sequences, the  
 5345 gradient of the conditional log-likelihood is the difference between the observed and ex-  
 5346 pected counts of each feature. The expectation  $E_{\tau|\mathbf{w}}[\mathbf{f}(\tau, \mathbf{w}^{(i)}); \boldsymbol{\theta}]$  requires summing over  
 5347 all possible parses, and computing the marginal probabilities of anchored productions,  
 5348  $p(X \rightarrow \alpha, (i, j, k) | \mathbf{w})$ . In CRF sequence labeling, marginal probabilities over tag bigrams  
 5349 are computed by the two-pass **forward-backward algorithm** (§ 7.5.3.3). The analogue for  
 5350 context-free grammars is the **inside-outside algorithm**, in which marginal probabilities  
 5351 are computed from terms generated by an upward and downward pass over the parsing  
 5352 chart:

Figure 10.3: The two cases faced by the outside recurrence in the computation of  $\beta(i, j, X)$ 

- The upward pass is performed by the **inside recurrence**, which is described in § 10.3.2. Each inside variable  $\alpha(i, j, X)$  is the score of deriving  $w_{i+1:j}$  from the non-terminal  $X$ . In a PCFG, this corresponds to the log-probability  $\log p(w_{i+1:j} \mid X)$ . This is computed by the recurrence,

$$\alpha(i, j, X) \triangleq \log \sum_{(X \rightarrow Y \ Z)} \sum_{k=i+1}^j \exp (\psi(X \rightarrow Y \ Z, (i, j, k)) + \alpha(i, k, Y) + \alpha(k, j, Z)). \quad [10.26]$$

5353        The initial condition of this recurrence is  $\alpha(m - 1, m, X) = \psi(X \rightarrow w_m)$ . The de-  
5354        nominator  $\sum_{\tau \in \mathcal{T}(w)} \exp \Psi(\tau)$  is equal to  $\exp \alpha(0, M, S)$ .

- The downward pass is performed by the **outside recurrence**, which recursively populates the same table structure, starting at the root of the tree. Each outside variable  $\beta(i, j, X)$  is the score of having a phrase of type  $X$  covering the span  $(i + 1 : j)$ , joint with the exterior context  $w_{1:i}$  and  $w_{j+1:M}$ . In a PCFG, this corresponds to the log probability  $\log p((X, i + 1, j), w_{1:i}, w_{j+1:M})$ . Each outside variable is computed by the recurrence,

$$\exp \beta(i, j, X) \triangleq \sum_{(Y \rightarrow X \ Z)} \sum_{k=j+1}^M \exp [\psi(Y \rightarrow X \ Z, (i, k, j)) + \alpha(j, k, Z) + \beta(i, k, Y)] \quad [10.27]$$

$$+ \sum_{(Y \rightarrow Z \ X)} \sum_{k=0}^{i-1} \exp [\psi(Y \rightarrow Z \ X, (k, i, j)) + \alpha(k, i, Z) + \beta(k, j, Y)]. \quad [10.28]$$

5355        The first line of Equation 10.28 is the score under the condition that  $X$  is a left child  
5356        of its parent, which spans  $w_{i+1:k}$ , with  $k > j$ ; the second line is the score under the  
5357        condition that  $X$  is a right child of its parent  $Y$ , which spans  $w_{k+1:j}$ , with  $k < i$ .  
5358        The two cases are shown in Figure 10.3. In each case, we sum over all possible  
5359        productions with  $X$  on the right-hand side. The parent  $Y$  is bounded on one side

5360 by either  $i$  or  $j$ , depending on whether  $X$  is a left or right child of  $Y$ ; we must sum  
 5361 over all possible values for the other boundary. The initial conditions for the outside  
 5362 recurrence are  $\beta(0, M, S) = 0$  and  $\beta(0, M, X \neq S) = -\infty$ .

The marginal probability of a non-terminal  $X$  over span  $w_{i+1:j}$  is written  $p(X \rightsquigarrow w_{i+1:j} | w)$ , and can be computed from the inside and outside scores,

$$p(X \rightsquigarrow w_{i+1:j} | w) = \frac{p(X \rightsquigarrow w_{i+1:j}, w)}{p(w)} \quad [10.29]$$

$$= \frac{p(w_{i+1:j} | X) \times p(X, w_{1:i}, w_{j+1:M})}{p(w)} \quad [10.30]$$

$$= \frac{\exp(\alpha(i, j, X) + \beta(i, j, X))}{\exp \alpha(0, M, S)}. \quad [10.31]$$

5363 Marginal probabilities of individual productions can be computed similarly (see exercise  
 5364 2). These marginal probabilities can be used for training a conditional random field parser,  
 5365 and also for the task of unsupervised **grammar induction**, in which a PCFG is estimated  
 5366 from a dataset of unlabeled text (Lari and Young, 1990; Pereira and Schabes, 1992).

#### 5367 10.4.4 Neural context-free grammars

5368 Recent work has applied neural representations to parsing, representing each span with  
 5369 a dense numerical vector (Socher et al., 2013; Durrett and Klein, 2015; Cross and Huang,  
 5370 2016).<sup>4</sup> For example, the anchor  $(i, j, k)$  and sentence  $w$  can be associated with a fixed-  
 5371 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),

$$\tilde{\mathbf{v}}_{(i,j,k)} = \text{FeedForward}(\mathbf{v}_{(i,j,k)}) \quad [10.33]$$

$$\psi(X \rightarrow \alpha, (i, j, k)) = \tilde{\mathbf{v}}_{(i,j,k)}^\top \Theta \mathbf{f}(X \rightarrow \alpha), \quad [10.34]$$

5372 where  $\mathbf{f}(X \rightarrow \alpha)$  is a vector of discrete features of the production, and  $\Theta$  is a parameter  
 5373 matrix. The matrix  $\Theta$  and the parameters of the feedforward network can be learned by  
 5374 backpropagating from an objective such as the margin loss or the negative conditional  
 5375 log-likelihood.

---

<sup>4</sup>Earlier 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).

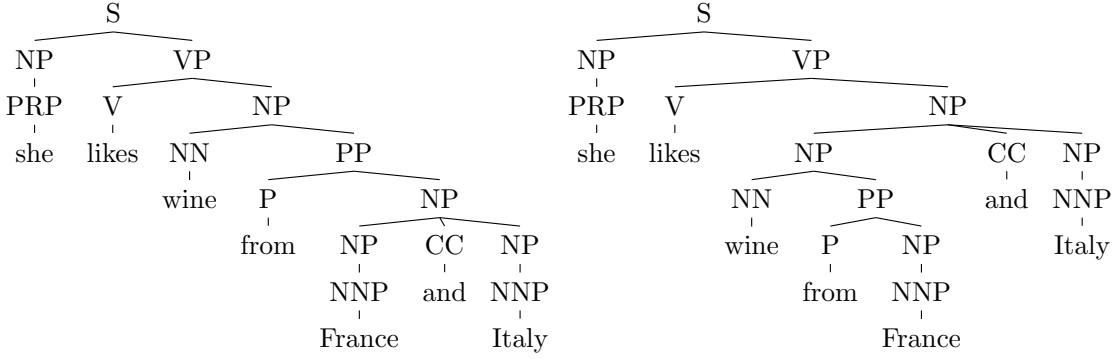


Figure 10.4: The left parse is preferable because of the conjunction of phrases headed by *France* and *Italy*, but these parses cannot be distinguished by a WCFG.

## 5376 10.5 Grammar refinement

5377 The locality assumptions underlying CFG parsing depend on the granularity of the non-  
 5378 terminals. For the Penn Treebank non-terminals, there are several reasons to believe that  
 5379 these assumptions are too strong to enable accurate parsing (Johnson, 1998):

- 5380 • The context-free assumption is too strict: for example, the probability of the produc-  
 5381 tion  $NP \rightarrow NP\ PP$  is much higher (in the PTB) if the parent of the noun phrase is a  
 5382 verb phrase (indicating that the NP is a direct object) than if the parent is a sentence  
 5383 (indicating that the NP is the subject of the sentence).
- 5384 • The Penn Treebank non-terminals are too coarse: there are many kinds of noun  
 5385 phrases and verb phrases, and accurate parsing sometimes requires knowing the  
 5386 difference. As we have already seen, when faced with prepositional phrase at-  
 5387 tachment ambiguity, a weighted CFG will either always choose NP attachment (if  
 5388  $\psi(NP \rightarrow NP\ PP) > \psi(VP \rightarrow VP\ PP)$ ), or it will always choose VP attachment. To  
 5389 get more nuanced behavior, more fine-grained non-terminals are needed.
- 5390 • More generally, accurate parsing requires some amount of **semantics** — understand-  
 5391 ing the meaning of the text to be parsed. Consider the example *cats scratch people with*  
 5392 *claws*: knowledge of about *cats*, *claws*, and scratching is necessary to correctly resolve  
 5393 the attachment ambiguity.

5394 An extreme example is shown in Figure 10.4. The analysis on the left is preferred  
 5395 because of the conjunction of similar entities *France* and *Italy*. But given the non-terminals  
 5396 shown in the analyses, there is no way to differentiate these two parses, since they include  
 5397 exactly the same productions. What is needed seems to be more precise non-terminals.  
 5398 One possibility would be to rethink the linguistics behind the Penn Treebank, and ask

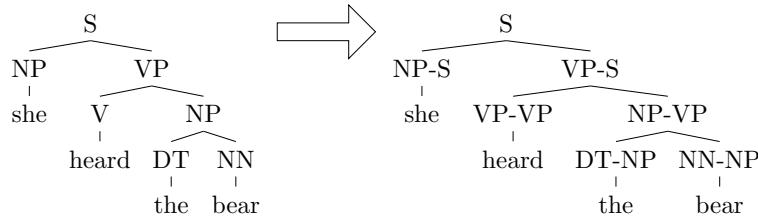


Figure 10.5: Parent annotation in a CFG derivation

5399 the annotators to try again. But the original annotation effort took five years, and there  
 5400 is a little appetite for another annotation effort of this scope. Researchers have therefore  
 5401 turned to automated techniques.

### 5402 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):

$$\Pr(\text{NP} \rightarrow \text{NP PP}) = 11\% \quad [10.35]$$

$$\Pr(\text{NP under S} \rightarrow \text{NP PP}) = 9\% \quad [10.36]$$

$$\Pr(\text{NP under VP} \rightarrow \text{NP PP}) = 23\%. \quad [10.37]$$

5403 This phenomenon can be captured by **parent annotation** (Johnson, 1998), in which each  
 5404 non-terminal is augmented with the identity of its parent, as shown in Figure 10.5). This is  
 5405 sometimes called **vertical Markovization**, since a Markov dependency is introduced be-  
 5406 tween each node and its parent (Klein and Manning, 2003). It is analogous to moving from  
 5407 a bigram to a trigram context in a hidden Markov model. In principle, parent annotation  
 5408 squares the size of the set of non-terminals, which could make parsing considerably less  
 5409 efficient. But in practice, the increase in the number of non-terminals that actually appear  
 5410 in the data is relatively modest (Johnson, 1998).

5411 Parent annotation weakens the WCFG locality assumptions. This improves accuracy  
 5412 by enabling the parser to make more fine-grained distinctions, which better capture real  
 5413 linguistic phenomena. However, each production is more rare, and so careful smoothing  
 5414 or regularization is required to control the variance over production scores.

### 5415 10.5.2 Lexicalized context-free grammars

5416 The examples in § 10.2.2 demonstrate the importance of individual words in resolving  
 5417 parsing ambiguity: the preposition *on* is more likely to attach to *met*, while the preposition  
 5418 *of* is more likely to attachment to *President*. But of all word pairs, which are relevant to  
 5419 attachment decisions? Consider the following variants on the original examples:

- 5420 (10.4) We met the President of Mexico.
- 5421 (10.5) We met the first female President of Mexico.
- 5422 (10.6) They had supposedly met the President on Monday.

5423 The underlined words are the **head words** of their respective phrases: *met* heads the verb  
 5424 phrase, and *President* heads the direct object noun phrase. These heads provide useful  
 5425 semantic information. But they break the context-free assumption, which states that the  
 5426 score for a production depends only on the parent and its immediate children, and not  
 5427 the substructure under each child.

The incorporation of head words into context-free parsing is known as **lexicalization**,  
 and is implemented in rules of the form,

$$\text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(of) \quad [10.38]$$

$$\text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(on). \quad [10.39]$$

5428 Lexicalization was a major step towards accurate PCFG parsing. It requires solving three  
 5429 problems: identifying the heads of all constituents in a treebank; parsing efficiently while  
 5430 keeping track of the heads; and estimating the scores for lexicalized productions.

#### 5431 10.5.2.1 Identifying head words

5432 The head of a constituent is the word that is the most useful for determining how that  
 5433 constituent is integrated into the rest of the sentence.<sup>5</sup> The head word of a constituent is  
 5434 determined recursively: for any non-terminal production, the head of the left-hand side  
 5435 must be the head of one of the children. The head is typically selected according to a set of  
 5436 deterministic rules, sometimes called **head percolation rules**. In many cases, these rules  
 5437 are straightforward: the head of a noun phrase in a  $\text{NP} \rightarrow \text{DET NN}$  production is the head

---

<sup>5</sup>This is a pragmatic definition, befitting our goal of using head words to improve parsing; for a more formal definition, see (Bender, 2013, chapter 7).

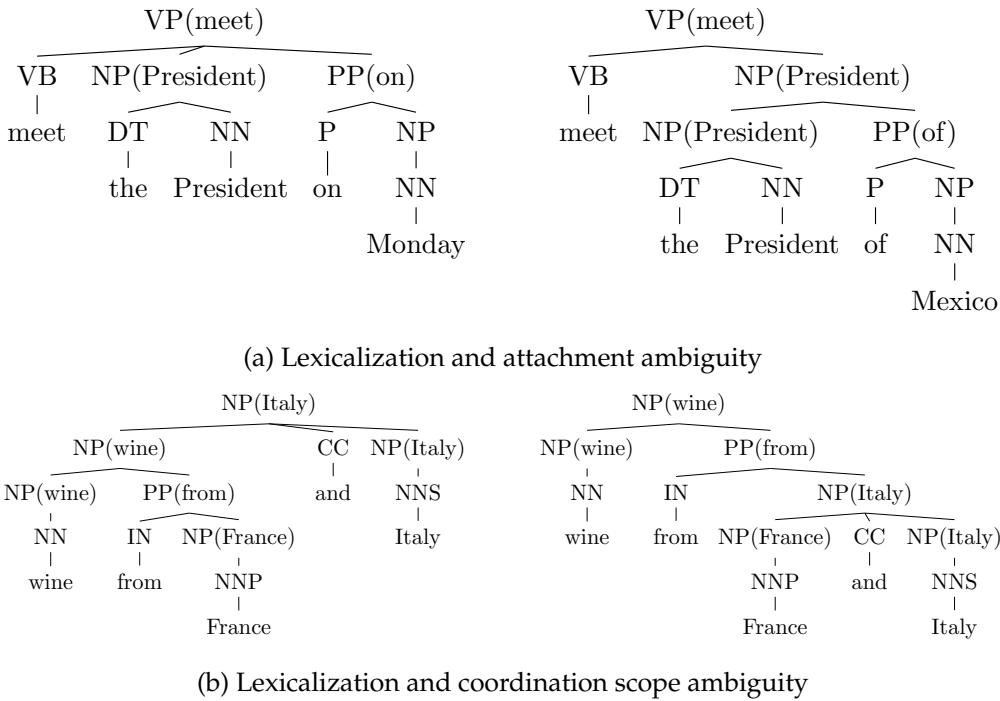


Figure 10.6: Examples of lexicalization

5438 of the noun; the head of a sentence in a  $S \rightarrow NP VP$  production is the head of the verb  
 5439 phrase.

5440 Table 10.3 shows a fragment of the head percolation rules used in many English pars-  
 5441 ing systems. The meaning of the first rule is that to find the head of an  $S$  constituent, first  
 5442 look for the rightmost  $VP$  child; if you don't find one, then look for the rightmost  $SBAR$   
 5443 child, and so on down the list. Verb phrases are headed by left verbs (the head of *can plan*  
 5444 *on walking* is *planned*, since the modal verb *can* is tagged *MD*); noun phrases are headed by  
 5445 the rightmost noun-like non-terminal (so the head of *the red cat* is *cat*),<sup>6</sup> and prepositional  
 5446 phrases are headed by the preposition (the head of *at Georgia Tech* is *at*). Some of these  
 5447 rules are somewhat arbitrary — there's no particular reason why the head of *cats and dogs*  
 5448 should be *dogs* — but the point here is just to get some lexical information that can support  
 5449 parsing, not to make deep claims about syntax. Figure 10.6 shows the application of these  
 5450 rules to two of the running examples.

<sup>6</sup>The 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>

### 5451 10.5.2.2 Parsing lexicalized context-free grammars

5452 A naïve application of lexicalization would simply increase the set of non-terminals by  
 5453 taking the cross-product with the set of terminal symbols, so that the non-terminals now  
 5454 include symbols like  $NP(President)$  and  $VP(meet)$ . Under this approach, the CKY parsing  
 5455 algorithm could be applied directly to the lexicalized production rules. However, the  
 5456 complexity would be cubic in the size of the vocabulary of terminal symbols, which would  
 5457 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  $t_\ell$  and  $t_r$  respectively, leading to the following recurrence:

$$t_\ell[i, j, h, X] = \max_{(X \rightarrow Y Z)} \max_{k > h} \max_{k < h' \leq j} t[i, k, h, Y] + t[k, j, h', Z] + \psi(X(h) \rightarrow Y(h)Z(h')) \quad [10.40]$$

$$t_r[i, j, h, X] = \max_{(X \rightarrow Y Z)} \max_{k < h} \max_{i < h' \leq k} t[i, k, h', Y] + t[k, j, h, Z] + (\psi(X(h) \rightarrow Y(h')Z(h))) \quad [10.41]$$

$$t[i, j, h, X] = \max(t_\ell[i, j, h, X], t_r[i, j, h, X]). \quad [10.42]$$

5458 To compute  $t_\ell$ , we maximize over all split points  $k > h$ , since the head word must be in  
 5459 the left child. We then maximize again over possible head words  $h'$  for the right child. An  
 5460 analogous computation is performed for  $t_r$ . The size of the table is now  $\mathcal{O}(M^3N)$ , where  
 5461  $M$  is the length of the input and  $N$  is the number of non-terminals. Furthermore, each  
 5462 cell is computed by performing  $\mathcal{O}(M^2)$  operations, since we maximize over both the split  
 5463 point  $k$  and the head  $h'$ . The time complexity of the algorithm is therefore  $\mathcal{O}(RM^5N)$ ,  
 5464 where  $R$  is the number of rules in the grammar. Fortunately, more efficient solutions are  
 5465 possible. In general, the complexity of parsing can be reduced to  $\mathcal{O}(M^4)$  in the length of

5466 the input; for a broad class of lexicalized CFGs, the complexity can be made cubic in the  
 5467 length of the input, just as in unlexicalized CFGs (Eisner, 2000).

5468 **10.5.2.3 Estimating lexicalized context-free grammars**

5469 The final problem for lexicalized parsing is how to estimate weights for lexicalized pro-  
 5470 ductions  $X(i) \rightarrow Y(j) Z(k)$ . These productions are said to be **bilexical**, because they  
 5471 involve scores over pairs of words: in the example *meet the President of Mexico*, we hope  
 5472 to choose the correct attachment point by modeling the bilexical affinities of (*meet, of*) and  
 5473 (*President, of*). The number of such word pairs is quadratic in the size of the vocabulary,  
 5474 making it difficult to estimate the weights of lexicalized production rules directly from  
 5475 data. This is especially true for probabilistic context-free grammars, in which the weights  
 5476 are obtained from smoothed relative frequency. In a treebank with a million tokens, a  
 5477 vanishingly small fraction of the possible lexicalized productions will be observed more  
 5478 than once.<sup>7</sup> The Charniak (1997) and Collins (1997) parsers therefore focus on approxi-  
 5479 mating the probabilities of lexicalized productions, using various smoothing techniques  
 5480 and independence assumptions.

In discriminatively-trained weighted context-free grammars, the scores for each production can be computed from a set of features, which can be made progressively more fine-grained (Finkel et al., 2008). For example, the score of the lexicalized production  $\text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(of)$  can be computed from the following features:

$$\begin{aligned} f(\text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(of)) = & \{\text{NP}(*) \rightarrow \text{NP}(*) \text{ PP}(*), \\ & \text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(*), \\ & \text{NP}(*) \rightarrow \text{NP}(*) \text{ PP}(of), \\ & \text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(of)\} \end{aligned}$$

5481 The first feature scores the unlexicalized production  $\text{NP} \rightarrow \text{NP PP}$ ; the next two features  
 5482 lexicalize only one element of the production, thereby scoring the appropriateness of  $\text{NP}$   
 5483 attachment for the individual words *President* and *of*; the final feature scores the specific  
 5484 bilexical affinity of *President* and *of*. For bilexical pairs that are encountered frequently in  
 5485 the treebank, this bilexical feature can play an important role in parsing; for pairs that are  
 5486 absent or rare, regularization will drive its weight to zero, forcing the parser to rely on the  
 5487 more coarse-grained features.

5488 In chapter 14, we will encounter techniques for clustering words based on their **distribu-**  
 5489 **tional** properties — the contexts in which they appear. Such a clustering would group  
 5490 rare and common words, such as *whale*, *shark*, *Leviathan*. Word clusters can be used

---

<sup>7</sup>The 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).

5491 as features in discriminative lexicalized parsing, striking a middle ground between full  
 5492 lexicalization and non-terminals (Finkel et al., 2008). In this way, labeled examples con-  
 5493 taining relatively common words like *whale* can help to improve parsing for rare words  
 5494 like *beluga*, as long as those two words are clustered together.

5495 **10.5.3 \*Refinement grammars**

5496 Lexicalization improves on context-free parsing by adding detailed information in the  
 5497 form of lexical heads. However, estimating the scores of lexicalized productions is dif-  
 5498 ficult. Klein and Manning (2003) argue that the right level of linguistic detail is some-  
 5499 where between treebank categories and individual words. Some parts-of-speech and non-  
 5500 terminals are truly substitutable: for example, *cat*/N and *dog*/N. But others are not: for  
 5501 example, the preposition *of* exclusively attaches to nouns, while the preposition *as* is more  
 5502 likely to modify verb phrases. Klein and Manning (2003) obtained a 2% improvement in  
 5503 *F*-MEASURE on a parent-annotated PCFG parser by making a single change: splitting the  
 5504 preposition category into six subtypes. They propose a series of linguistically-motivated  
 5505 refinements to the Penn Treebank annotations, which in total yielded a 40% error reduc-  
 5506 tion.

5507 Non-terminal refinement process can be automated by treating the refined categories  
 5508 as latent variables. For example, we might split the noun phrase non-terminal into NP1, NP2, NP3, ...,  
 5509 without defining in advance what each refined non-terminal corresponds to. This can  
 5510 be treated as **partially supervised learning**, similar to the multi-component document  
 5511 classification model described in § 5.2.3. A latent variable PCFG can be estimated by  
 5512 expectation-maximization (Matsuzaki et al., 2005):

- 5513 • In the E-step, estimate a marginal distribution  $q$  over the refinement type of each  
 5514 non-terminal in each derivation. These marginals are constrained by the original  
 5515 annotation: an NP can be reannotated as NP4, but not as VP3. Marginal probabili-  
 5516 ties over refined productions can be computed from the **inside-outside algorithm**,  
 5517 as described in § 10.4.3, where the E-step enforces the constraints imposed by the  
 5518 original annotations.
- 5519 • In the M-step, recompute the parameters of the grammar, by summing over the  
 5520 probabilities of anchored productions that were computed in the E-step:

$$E[\text{count}(X \rightarrow Y Z)] = \sum_{i=0}^M \sum_{j=i}^M \sum_{k=i}^j p(X \rightarrow Y Z, (i, j, k) | \mathbf{w}). \quad [10.43]$$

5521 As usual, this process can be iterated to convergence. To determine the number of re-  
 5522 finement types for each tag, Petrov et al. (2006) apply a split-merge heuristic; Liang et al.  
 5523 (2007) and Finkel et al. (2007) apply **Bayesian nonparametrics** (Cohen, 2016).

Proper nouns			
NNP-14	<i>Oct.</i>	<i>Nov.</i>	<i>Sept.</i>
NNP-12	<i>John</i>	<i>Robert</i>	<i>James</i>
NNP-2	<i>J.</i>	<i>E.</i>	<i>L.</i>
NNP-1	<i>Bush</i>	<i>Noriega</i>	<i>Peters</i>
NNP-15	<i>New</i>	<i>San</i>	<i>Wall</i>
NNP-3	<i>York</i>	<i>Francisco</i>	<i>Street</i>
Personal Pronouns			
PRP-0	<i>It</i>	<i>He</i>	<i>I</i>
PRP-1	<i>it</i>	<i>he</i>	<i>they</i>
PRP-2	<i>it</i>	<i>them</i>	<i>him</i>

Table 10.4: Examples of automatically refined non-terminals and some of the words that they generate (Petrov et al., 2006).

5524 Some examples of refined non-terminals are shown in Table 10.4. The proper nouns  
 5525 differentiate months, first names, middle initials, last names, first names of places, and  
 5526 second names of places; each of these will tend to appear in different parts of grammatical  
 5527 productions. The personal pronouns differentiate grammatical role, with PRP-0 appear-  
 5528 ing in subject position at the beginning of the sentence (note the capitalization), PRP-1  
 5529 appearing in subject position but not at the beginning of the sentence, and PRP-2 appear-  
 5530 ing in object position.

## 5531 10.6 Beyond context-free parsing

5532 In the context-free setting, the score for a parse is a combination of the scores of individual  
 5533 productions. As we have seen, these models can be improved by using finer-grained non-  
 5534 terminals, via parent-annotation, lexicalization, and automated refinement. However, the  
 5535 inherent limitations to the expressiveness of context-free parsing motivate the consider-  
 5536 ation of other search strategies. These strategies abandon the optimality guaranteed by  
 5537 bottom-up parsing, in exchange for the freedom to consider arbitrary properties of the  
 5538 proposed parses.

### 5539 10.6.1 Reranking

5540 A simple way to relax the restrictions of context-free parsing is to perform a two-stage pro-  
 5541 cess, in which a context-free parser generates a  $k$ -best list of candidates, and a **reranker**  
 5542 then selects the best parse from this list (Charniak and Johnson, 2005; Collins and Koo,  
 5543 2005). The reranker can be trained from an objective that is similar to multi-class classi-  
 5544 fication: the goal is to learn weights that assign a high score to the reference parse, or to

5545 the parse on the  $k$ -best list that has the lowest error. In either case, the reranker need only  
 5546 evaluate the  $K$  best parses, and so no context-free assumptions are necessary. This opens  
 5547 the door to more expressive scoring functions:

- 5548 • It is possible to incorporate arbitrary non-local features, such as the structural par-  
 5549 allelism and right-branching orientation of the parse (Charniak and Johnson, 2005).  
 5550 • Reranking enables the use of **recursive neural networks**, in which each constituent  
 5551 span  $w_{i+1:j}$  receives a vector  $\mathbf{u}_{i,j}$  which is computed from the vector representa-  
 5552 tions of its children, using a composition function that is linked to the production  
 5553 rule (Socher et al., 2013), e.g.,

$$\mathbf{u}_{i,j} = f \left( \Theta_{X \rightarrow Y} Z \begin{bmatrix} \mathbf{u}_{i,k} \\ \mathbf{u}_{k,j} \end{bmatrix} \right) \quad [10.44]$$

5554 The overall score of the parse can then be computed from the final vector,  $\Psi(\tau) =$   
 5555  $\theta \mathbf{u}_{0,M}$ .

5556 Reranking can yield substantial improvements in accuracy. The main limitation is that it  
 5557 can only find the best parse among the  $K$ -best offered by the generator, so it is inherently  
 5558 limited by the ability of the bottom-up parser to find high-quality candidates.

### 5559 10.6.2 Transition-based parsing

5560 Structure prediction can be viewed as a form of search. An alternative to bottom-up pars-  
 5561 ing is to read the input from left-to-right, gradually building up a parse structure through  
 5562 a series of **transitions**. Transition-based parsing is described in more detail in the next  
 5563 chapter, in the context of dependency parsing. However, it can also be applied to CFG  
 5564 parsing, as briefly described here.

5565 For any context-free grammar, there is an equivalent **pushdown automaton**, a model  
 5566 of computation that accepts exactly those strings that can be derived from the grammar.  
 5567 This computational model consumes the input from left to right, while pushing and pop-  
 5568 ping elements on a stack. This architecture provides a natural transition-based parsing  
 5569 framework for context-free grammars, known as **shift-reduce parsing**.

5570 Shift-reduce parsing is a type of transition-based parsing, in which the parser can take  
 5571 the following actions:

- 5572 • *shift* the next terminal symbol onto the stack;  
 5573 • *unary-reduce* the top item on the stack, using a unary production rule in the gram-  
 5574 mar;  
 5575 • *binary-reduce* the top two items onto the stack, using a binary production rule in the  
 5576 grammar.

5577 The set of available actions is constrained by the situation: the parser can only shift if  
 5578 there are remaining terminal symbols in the input, and it can only reduce if an applicable  
 5579 production rule exists in the grammar. If the parser arrives at a state where the input  
 5580 has been completely consumed, and the stack contains only the element S, then the input  
 5581 is accepted. If the parser arrives at a non-accepting state where there are no possible  
 5582 actions, the input is rejected. A parse error occurs if there is some action sequence that  
 5583 would accept an input, but the parser does not find it.

5584 **Example** Consider the input *we eat sushi* and the grammar in Table 10.1. The input can  
 5585 be parsed through the following sequence of actions:

- 5586 1. **Shift** the first token *we* onto the stack.
- 5587 2. **Reduce** the top item on the stack to NP, using the production  $NP \rightarrow we$ .
- 5588 3. **Shift** the next token *eat* onto the stack, and **reduce** it to V with the production  $V \rightarrow$   
 5589 *eat*.
- 5590 4. **Shift** the final token *sushi* onto the stack, and **reduce** it to NP. The input has been  
 5591 completely consumed, and the stack contains [NP, V, NP].
- 5592 5. **Reduce** the top two items using the production  $VP \rightarrow V NP$ . The stack now con-  
 5593 tains [VP, NP].
- 5594 6. **Reduce** the top two items using the production  $S \rightarrow NP VP$ . The stack now contains  
 5595 [S]. Since the input is empty, this is an accepting state.

5596 One thing to notice from this example is that the number of shift actions is equal to the  
 5597 length of the input. The number of reduce actions is equal to the number of non-terminals  
 5598 in the analysis, which grows linearly in the length of the input. Thus, the overall time  
 5599 complexity of shift-reduce parsing is linear in the length of the input (assuming the com-  
 5600 plexity of each individual classification decision is constant in the length of the input).  
 5601 This is far better than the cubic time complexity required by CKY parsing.

5602 **Transition-based parsing as inference** In general, it is not possible to guarantee that  
 5603 a transition-based parser will find the optimal parse,  $\text{argmax}_\tau \Psi(\tau; \mathbf{w})$ , even under the  
 5604 usual CFG independence assumptions. We could assign a score to each anchored parsing  
 5605 action in each context, with  $\psi(a, c)$  indicating the score of performing action  $a$  in context  $c$ .  
 5606 One might imagine that transition-based parsing could efficiently find the derivation that  
 5607 maximizes the sum of such scores. But this too would require backtracking and searching  
 5608 over an exponentially large number of possible action sequences: if a bad decision is  
 5609 made at the beginning of the derivation, then it may be impossible to recover the optimal  
 5610 action sequence without backtracking to that early mistake. This is known as a **search**  
 5611 **error**. Transition-based parsers can incorporate arbitrary features, without the restrictive

5612 independence assumptions required by chart parsing; search errors are the price that must  
 5613 be paid for this flexibility.

5614 **Learning transition-based parsing** Transition-based parsing can be combined with ma-  
 5615 chine learning by training a classifier to select the correct action in each situation. This  
 5616 classifier is free to choose any feature of the input, the state of the parser, and the parse  
 5617 history. However, there is no optimality guarantee: the parser may choose a suboptimal  
 5618 parse, due to a mistake at the beginning of the analysis. Nonetheless, some of the strongest  
 5619 CFG parsers are based on the shift-reduce architecture, rather than CKY. A recent genera-  
 5620 tion of models links shift-reduce parsing with recurrent neural networks, updating a  
 5621 hidden state vector while consuming the input (e.g., Cross and Huang, 2016; Dyer et al.,  
 5622 2016). Learning algorithms for transition-based parsing are discussed in more detail in  
 5623 § 11.3.

5624 **Exercises**

1. Consider the following PCFG:

$$p(X \rightarrow X X) = \frac{1}{2} \quad [10.45]$$

$$p(X \rightarrow Y) = \frac{1}{2} \quad [10.46]$$

$$p(Y \rightarrow \sigma) = \frac{1}{|\Sigma|}, \forall \sigma \in \Sigma \quad [10.47]$$

5625 a) Compute the probability  $p(\hat{\tau})$  of the maximum probability parse for a string  
 5626  $w \in \Sigma^M$ .

5627 b) Compute the marginal probability  $p(w) = \sum_{\tau: \text{yield}(\tau)=w} p(\tau)$ .

5628 c) Compute the conditional probability  $p(\hat{\tau} | w)$ .

- 5629 2. Use the inside and outside scores to compute the marginal probability  $p(X_{i:j} \rightarrow Y_{i:k-1} Z_{k:j} | w)$ ,  
 5630 indicating that  $Y$  spans  $w_{i:k-1}$ ,  $Z$  spans  $w_{k:j}$ , and  $X$  is the parent of  $Y$  and  $Z$ , span-  
 5631 ning  $w_{i:j}$ .
- 5632 3. Suppose that the potentials  $\Psi(X \rightarrow \alpha)$  are log-probabilities, so that  $\sum_{\alpha} \exp \Psi(X \rightarrow \alpha) = 1$   
 5633 for all  $X$ . Verify that the semiring inside recurrence from Equation 10.26 generates  
 5634 the log-probability  $\log p(w) = \log \sum_{\tau: \text{yield}(\tau)=w} p(\tau)$ .
- 5635 4. more exercises tk

# 5636 Chapter 11

## 5637 Dependency parsing

5638 The previous chapter discussed algorithms for analyzing sentences in terms of nested con-  
5639 stituents, such as noun phrases and verb phrases. However, many of the key sources of  
5640 ambiguity in phrase-structure analysis relate to questions of **attachment**: where to attach a  
5641 prepositional phrase or complement clause, how to scope a coordinating conjunction, and  
5642 so on. These attachment decisions can be represented with a more lightweight structure:  
5643 a directed graph over the words in the sentence, known as a **dependency parse**. Syntac-  
5644 tic annotation has shifted its focus to such dependency structures: at the time of this  
5645 writing, the **Universal Dependencies** project offers more than 100 dependency treebanks  
5646 for more than 60 languages.<sup>1</sup> This chapter will describe the linguistic ideas underlying  
5647 dependency grammar, and then discuss exact and transition-based parsing algorithms.  
5648 The chapter will also discuss recent research on **learning to search** in transition-based  
5649 structure prediction.

### 5650 11.1 Dependency grammar

5651 While **dependency grammar** has a rich history of its own (Tesnière, 1966; Kübler et al.,  
5652 2009), it can be motivated by extension from the lexicalized context-free grammars that  
5653 we encountered in previous chapter (§ 10.5.2). Recall that lexicalization augments each  
5654 non-terminal with a **head word**. The head of a constituent is identified recursively, using  
5655 a set of **head rules**, as shown in Table 10.3. An example of a lexicalized context-free parse  
5656 is shown in Figure 11.1a. In this sentence, the head of the S constituent is the main verb,  
5657 *scratch*; this non-terminal then produces the noun phrase *the cats*, whose head word is  
5658 *cats*, and from which we finally derive the word *the*. Thus, the word *scratch* occupies the  
5659 central position for the sentence, with the word *cats* playing a supporting role. In turn, *cats*

---

<sup>1</sup>[universaldependencies.org](http://universaldependencies.org)

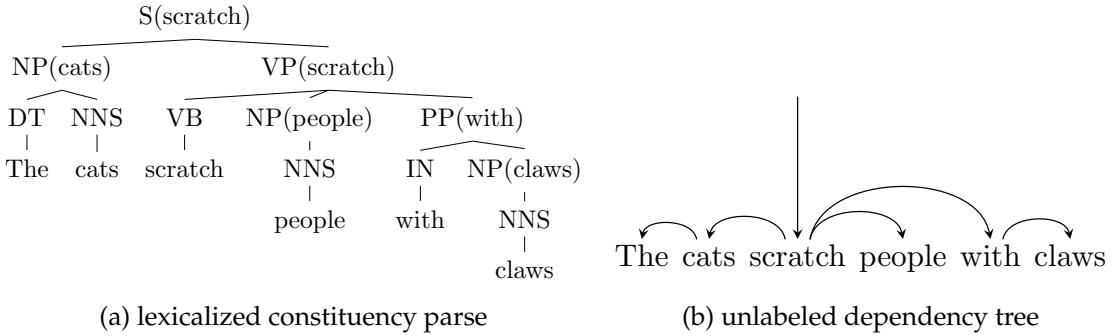


Figure 11.1: Dependency grammar is closely linked to lexicalized context free grammars: each lexical head has a dependency path to every other word in the constituent. (This example is based on the lexicalization rules from § 10.5.2, which make the preposition the head of a prepositional phrase. In the more contemporary Universal Dependencies annotations, the head of *with claws* would be *claws*, so there would be an edge *scratch* → *claws*.)

5660 occupies the central position for the noun phrase, with the word *the* playing a supporting  
5661 role.

5662 The relationships between words in a sentence can be formalized in a directed graph,  
5663 based on the lexicalized phrase-structure parse: create an edge  $(i, j)$  iff word  $i$  is the head  
5664 of a phrase whose child is a phrase headed by word  $j$ . Thus, in our example, we would  
5665 have *scratch* → *cats* and *cats* → *the*. We would not have the edge *scratch* → *the*, because  
5666 although  $S(\text{scratch})$  dominates  $\text{DET}(\text{the})$  in the phrase-structure parse tree, it is not its im-  
5667 mediate parent. These edges describe **syntactic dependencies**, a bilexical relationship  
5668 between a **head** and a **dependent**, which is at the heart of dependency grammar.

5669 Continuing to build out this **dependency graph**, we will eventually reach every word  
5670 in the sentence, as shown in Figure 11.1b. In this graph — and in all graphs constructed  
5671 in this way — every word has exactly one incoming edge, except for the root word, which  
5672 is indicated by a special incoming arrow from above. Furthermore, the graph is *weakly*  
5673 *connected*: if the directed edges were replaced with undirected edges, there would be a  
5674 path between all pairs of nodes. From these properties, it can be shown that there are no  
5675 cycles in the graph (or else at least one node would have to have more than one incoming  
5676 edge), and therefore, the graph is a tree. Because the graph includes all vertices, it is a  
5677 **spanning tree**.

### 5678 11.1.1 Heads and dependents

5679 A dependency edge implies an asymmetric syntactic relationship between the head and  
5680 dependent words, sometimes called **modifiers**. For a pair like *the cats* or *cats scratch*, how

5681 do we decide which is the head? Here are some possible criteria:

- 5682 • The head sets the syntactic category of the construction: for example, nouns are the  
5683 heads of noun phrases, and verbs are the heads of verb phrases.
- 5684 • The modifier may be optional while the head is mandatory: for example, in the  
5685 sentence *cats scratch people with claws*, the subtrees *cats scratch* and *cats scratch people*  
5686 are grammatical sentences, but *with claws* is not.
- 5687 • The head determines the morphological form of the modifier: for example, in lan-  
5688 guages that require gender agreement, the gender of the noun determines the gen-  
5689 der of the adjectives and determiners.
- 5690 • Edges should first connect content words, and then connect function words.

5691 As always, these guidelines sometimes conflict. The Universal Dependencies (UD)  
5692 project has attempted to identify a set of principles that can be applied to dozens of dif-  
5693 ferent languages (Nivre et al., 2016).<sup>2</sup> These guidelines are based on the universal part-  
5694 of-speech tags from chapter 8. They differ somewhat from the head rules described in  
5695 § 10.5.2: for example, on the principle that dependencies should relate content words, the  
5696 prepositional phrase *with claws* would be headed by *claws*, resulting in an edge *scratch* →  
5697 *claws*, and another edge *claws* → *with*.

5698 One objection to dependency grammar is that not all syntactic relations are asymmet-  
5699 ric. Coordination is one of the most obvious examples (Popel et al., 2013): in the sentence,  
5700 *Abigail and Max like kimchi* (Figure 11.2), which word is the head of the coordinated noun  
5701 phrase *Abigail and Max*? Choosing either *Abigail* or *Max* seems arbitrary; fairness argues  
5702 for making *and* the head, but this seems like the least important word in the noun phrase,  
5703 and selecting it would violate the principle of linking content words first. The Universal  
5704 Dependencies annotation system arbitrarily chooses the left-most item as the head — in  
5705 this case, *Abigail* — and includes edges from this head to both *Max* and the coordinating  
5706 conjunction *and*. These edges are distinguished by the labels CONJ (for the thing begin  
5707 conjoined) and CC (for the coordinating conjunction). The labeling system is discussed  
5708 next.

### 5709 11.1.2 Labeled dependencies

5710 Edges may be **labeled** to indicate the nature of the syntactic relation that holds between  
5711 the two elements. For example, in Figure 11.2, the label NSUBJ on the edge from *like* to  
5712 *Abigail* indicates that the subtree headed by *Abigail* is the noun subject of the verb *like*;  
5713 similarly, the label OBJ on the edge from *like* to *kimchi* indicates that the subtree headed by

---

<sup>2</sup>The latest and most specific guidelines are available at [universaldependencies.org/guidelines.html](http://universaldependencies.org/guidelines.html)

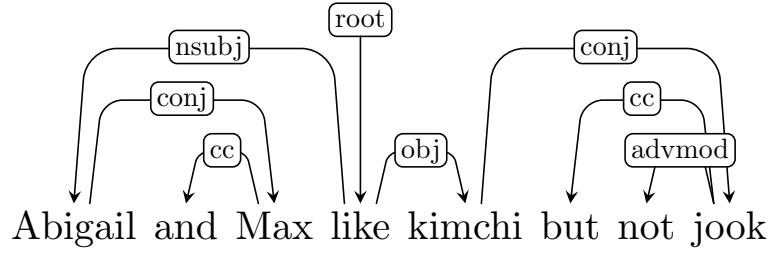


Figure 11.2: In the Universal Dependencies annotation system, the left-most item of a coordination is the head.

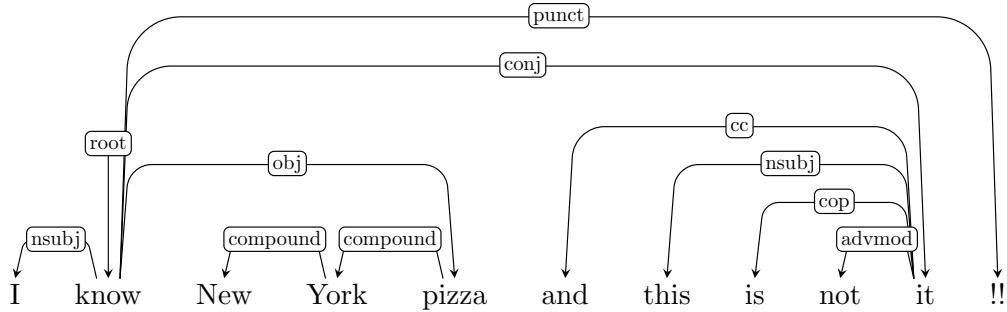


Figure 11.3: A labeled dependency parse from the English UD Treebank (reviews-361348-0006)

5714    *kimchi* is the object.<sup>3</sup> The negation *not* is treated as an adverbial modifier (ADVMOD) on  
5715    the noun *jook*.

5716    A slightly more complex example is shown in Figure 11.3. The multiword expression  
5717    *New York pizza* is treated as a “flat” unit of text, with the elements linked by the COM-  
5718    POUND relation. The sentence includes two clauses that are conjoined in the same way  
5719    that noun phrases are conjoined in Figure 11.2. The second clause contains a copula verb  
5720    (see § 8.1.1). For such clauses, we treat the “object” of the verb as the root — in this case,  
5721    *it* — and label the verb as a dependent, with the COP relation. This example also shows  
5722    how punctuations are treated, with label PUNCT.

### 5723    11.1.3 Dependency subtrees and constituents

5724    Dependency trees hide information that would be present in a CFG parse. Often what  
5725    is hidden is in fact irrelevant: for example, Figure 11.4 shows three different ways of

---

<sup>3</sup>Earlier work distinguished direct and indirect objects (De Marneffe and Manning, 2008), but this has been dropped in version 2.0 of the Universal Dependencies annotation system.

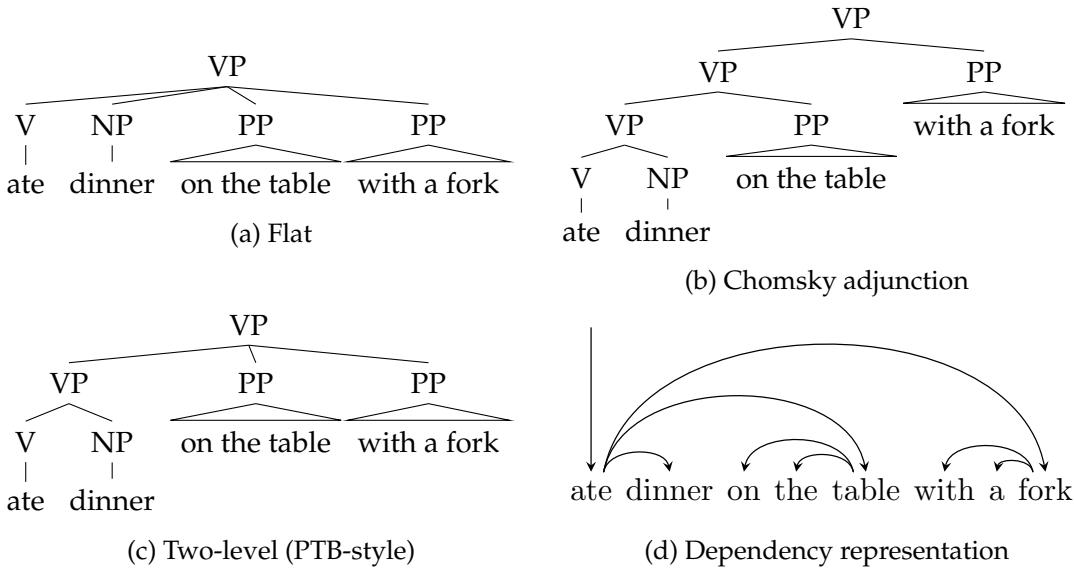


Figure 11.4: The three different CFG analyses of this verb phrase all correspond to a single dependency structure.

representing prepositional phrase adjuncts to the verb *ate*. Because there is apparently no meaningful difference between these analyses, the Penn Treebank decides by convention to use the two-level representation (see Johnson, 1998, for a discussion). As shown in Figure 11.4d, these three cases all look the same in a dependency parse.

But dependency grammar imposes its own set of annotation decisions, such as the identification of the head of a coordination (§ 11.1.1); without lexicalization, context-free grammar does not require either element in a coordination to be privileged in this way. Dependency parses can be disappointingly flat: for example, in the sentence *Yesterday, Abigail was reluctantly giving Max kimchi*, the root *giving* is the head of every dependency! The constituent parse arguably offers a more useful structural analysis for such cases.

**Projectivity** Thus far, we have defined dependency trees as spanning trees over a graph in which each word is a vertex. As we have seen, one way to construct such trees is by connecting the heads in a lexicalized constituent parse. However, there are spanning trees that cannot be constructed in this way. Syntactic constituents are *contiguous spans*. In a spanning tree constructed from a lexicalized constituent parse, the head  $h$  of any constituent that spans the nodes from  $i$  to  $j$  must have a path to every node in this span. This property is known as **projectivity**, and projective dependency parses are a restricted class of spanning trees. Informally, projectivity means that “crossing edges” are prohibited. The formal definition follows:

	% non-projective edges	% non-projective sentences
Czech	1.86%	22.42%
English	0.39%	7.63%
German	2.33%	28.19%

Table 11.1: Frequency of non-projective dependencies in three languages (Kuhlmann and Nivre, 2010)

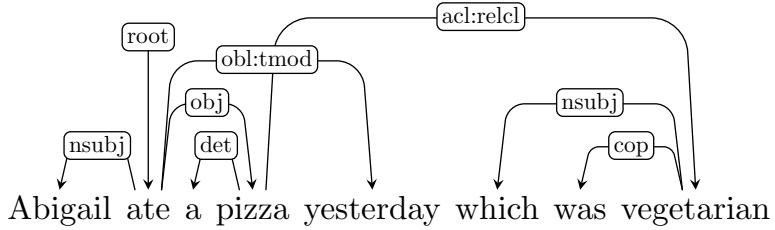


Figure 11.5: An example of a non-projective dependency parse. The “crossing edge” arises from the relative clause *which was vegetarian* and the oblique temporal modifier *yesterday*.

5745 **Definition 2** (Projectivity). *An edge from  $i$  to  $j$  is projective iff all  $k$  between  $i$  and  $j$  are descendants of  $i$ . A dependency parse is projective iff all its edges are projective.*

5747 Figure 11.5 gives an example of a non-projective dependency graph in English. This  
 5748 dependency graph does not correspond to any constituent parse. As shown in Table 11.1,  
 5749 non-projectivity is more common in languages such as Czech and German. Even though  
 5750 relatively few dependencies are non-projective in these languages, many sentences have  
 5751 at least one such dependency. As we will soon see, projectivity has important algorithmic  
 5752 consequences.

## 5753 11.2 Graph-based dependency parsing

5754 Let  $\mathbf{y} = \{i \xrightarrow{r} j\}$  represent a dependency graph, in which each edge is a relation  $r$  from  
 5755 head word  $i \in \{1, 2, \dots, M, \text{ROOT}\}$  to modifier  $j \in \{1, 2, \dots, M\}$ . The special node ROOT  
 5756 indicates the root of the graph, and  $M$  is the length of the input  $|\mathbf{w}|$ . Given a scoring  
 5757 function  $\Psi(\mathbf{y}, \mathbf{w}; \theta)$ , the optimal parse is,

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}(\mathbf{w})}{\operatorname{argmax}} \Psi(\mathbf{y}, \mathbf{w}; \theta), \quad [11.1]$$

5758 where  $\mathcal{Y}(\mathbf{w})$  is the set of valid dependency parses on the input  $\mathbf{w}$ . As usual, the number  
 5759 of possible labels  $|\mathcal{Y}(\mathbf{w})|$  is exponential in the length of the input (Wu and Chao, 2004).

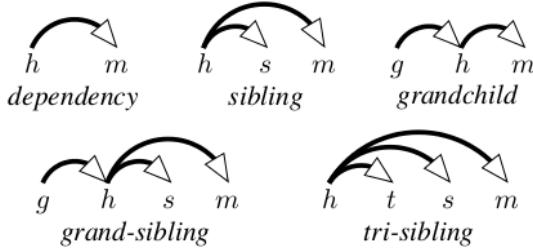


Figure 11.6: Feature templates for higher-order dependency parsing (Koo and Collins, 2010) [todo: permission]

5760 Algorithms that search over this space of possible graphs are known as **graph-based de-**  
 5761 **pendency parsers.**

In sequence labeling and constituent parsing, it was possible to search efficiently over an exponential space by choosing a feature function that decomposes into a sum of local feature vectors. A similar approach is possible for dependency parsing, by requiring the scoring function to decompose across dependency arcs  $i \rightarrow j$ :

$$\Psi(\mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}). \quad [11.2]$$

5762 Dependency parsers that operate under this assumption are known as **arc-factored**, since  
 5763 the overall score is a product of scores over all arcs.

**Higher-order dependency parsing** The arc-factored decomposition can be relaxed to allow higher-order dependencies. In **second-order dependency parsing**, the scoring function may include grandparents and siblings, as shown by the templates in Figure 11.6. The scoring function is,

$$\begin{aligned} \Psi(\mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) = & \sum_{i \xrightarrow{r} j \in \mathbf{y}} \sum_{k \xrightarrow{r'} i \in \mathbf{y}} \psi_{\text{grandparent}}(i \xrightarrow{r} j, k, r', \mathbf{w}; \boldsymbol{\theta}) \\ & \sum_{\substack{i \xrightarrow{r'} s \in \mathbf{y} \\ s \neq j}} \psi_{\text{sibling}}(i \xrightarrow{r} j, s, r', \mathbf{w}; \boldsymbol{\theta}). \end{aligned} \quad [11.3]$$

5764 The top line scores computes a scoring function that includes the grandparent  $k$ ; the  
 5765 bottom line computes a scoring function for each sibling  $s$ . For projective dependency  
 5766 graphs, there are efficient algorithms for second-order and third-order dependency pars-  
 5767 ing (Eisner, 1996; McDonald and Pereira, 2006; Koo and Collins, 2010); for non-projective  
 5768 dependency graphs, second-order dependency parsing is NP-hard (McDonald and Pereira,  
 5769 2006). The specific algorithms are discussed in the next section.

5770 **11.2.1 Graph-based parsing algorithms**

5771 The distinction between projective and non-projective dependency trees (§ 11.1.3) plays  
 5772 a key role in the choice of algorithms. Because projective dependency trees are closely  
 5773 related to (and can be derived from) lexicalized constituent trees, lexicalized parsing al-  
 5774 gorithms can be applied directly. For the more general problem of parsing to arbitrary  
 5775 spanning trees, a different class of algorithms is required. In both cases, arc-factored de-  
 5776 pendency parsing relies on precomputing the scores  $\psi(i \xrightarrow{r} j, w; \theta)$  for each potential  
 5777 edge. There are  $\mathcal{O}(M^2 R)$  such scores, where  $M$  is the length of the input and  $R$  is the  
 5778 number of dependency relation types, and this is a lower bound on the time and space  
 5779 complexity of any exact algorithm for arc-factored dependency parsing.

5780 **11.2.1.1 Projective dependency parsing**

5781 Any lexicalized constituency tree can be converted into a projective dependency tree by  
 5782 creating arcs between the heads of constituents and their parents, so any algorithm for  
 5783 lexicalized constituent parsing can be converted into an algorithm for projective depen-  
 5784 dency parsing, by converting arc scores into scores for lexicalized productions. As noted  
 5785 in § 10.5.2, there are cubic time algorithms for lexicalized constituent parsing, which are  
 5786 extensions of the CKY algorithm. Therefore, arc-factored projective dependency parsing  
 5787 can be performed in cubic time in the length of the input.

5788 Second-order projective dependency parsing can also be performed in cubic time, with  
 5789 minimal modifications to the lexicalized parsing algorithm (Eisner, 1996). It is possible to  
 5790 go even further, to **third-order dependency parsing**, in which the scoring function may  
 5791 consider great-grandparents, grand-siblings, and “tri-siblings”, as shown in Figure 11.6.  
 5792 Third-order dependency parsing can be performed in  $\mathcal{O}(M^4)$  time, which can be made  
 5793 practical through the use of pruning to eliminate unlikely edges (Koo and Collins, 2010).

5794 **11.2.1.2 Non-projective dependency parsing**

5795 In non-projective dependency parsing, the goal is to identify the highest-scoring span-  
 5796 ning tree over the words in the sentence. The arc-factored assumption ensures that the  
 5797 score for each spanning tree will be computed as a sum over scores for the edges, which  
 5798 are precomputed. Based on these scores, we build a weighted connected graph. Arc-  
 5799 factored non-projective dependency parsing is then equivalent to finding the spanning  
 5800 tree that achieves the maximum total score,  $\Psi(y, w) = \sum_{i \xrightarrow{r} j \in y} \psi(i \xrightarrow{r} j, w)$ . The **Chu-**  
 5801 **Liu-Edmonds algorithm** (Chu and Liu, 1965; Edmonds, 1967) computes this **maximum**  
 5802 **spanning tree** efficiently. It does this by first identifying the best incoming edge  $i \xrightarrow{r} j$  for  
 5803 each vertex  $j$ . If the resulting graph does not contain cycles, it is the maximum spanning  
 5804 tree. If there is a cycle, it is collapsed into a super-vertex, whose incoming and outgoing  
 5805 edges are based on the edges to the vertices in the cycle. The algorithm is then applied

5806 recursively to the resulting graph, and process repeats until a graph without cycles is  
 5807 obtained.

5808 The time complexity of identifying the best incoming edge for each vertex is  $\mathcal{O}(M^2R)$ ,  
 5809 where  $M$  is the length of the input and  $R$  is the number of relations; in the worst case, the  
 5810 number of cycles is  $\mathcal{O}(M)$ . Therefore, the complexity of the Chu-Liu-Edmonds algorithm  
 5811 is  $\mathcal{O}(M^3R)$ . This complexity can be reduced to  $\mathcal{O}(M^2N)$  by storing the edge scores in a  
 5812 Fibonacci heap (Gabow et al., 1986). For more detail on graph-based parsing algorithms,  
 5813 see Eisner (1997) and Kübler et al. (2009).

5814 **Higher-order non-projective dependency parsing** Given the tractability of higher-order  
 5815 projective dependency parsing, you may be surprised to learn that non-projective second-  
 5816 order dependency parsing is NP-Hard. This can be proved by reduction from the vertex  
 5817 cover problem (Neuhaus and Bröker, 1997). A heuristic solution is to do projective pars-  
 5818 ing first, and then post-process the projective dependency parse to add non-projective  
 5819 edges (Nivre and Nilsson, 2005). More recent work has applied techniques for approxi-  
 5820 mate inference in graphical models, including belief propagation (Smith and Eisner, 2008),  
 5821 integer linear programming (Martins et al., 2009), variational inference (Martins et al.,  
 5822 2010), and Markov Chain Monte Carlo (Zhang et al., 2014).

### 5823 11.2.2 Computing scores for dependency arcs

The arc-factored scoring function  $\psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})$  can be defined in several ways:

$$\text{Linear} \quad \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \boldsymbol{\theta} \cdot \mathbf{f}(i \xrightarrow{r} j, \mathbf{w}) \quad [11.4]$$

$$\text{Neural} \quad \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \text{Feedforward}([\mathbf{u}_{w_i}; \mathbf{u}_{w_j}]; \boldsymbol{\theta}) \quad [11.5]$$

$$\text{Generative} \quad \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \log p(w_j, r | w_i). \quad [11.6]$$

#### 5824 11.2.2.1 Linear feature-based arc scores

5825 Linear models for dependency parsing incorporate many of the same features used in  
 5826 sequence labeling and discriminative constituent parsing. These include:

- 5827 • the length and direction of the arc;
- 5828 • the words  $w_i$  and  $w_j$  linked by the dependency relation;
- 5829 • the prefixes, suffixes, and parts-of-speech of these words;
- 5830 • the neighbors of the dependency arc,  $w_{i-1}, w_{i+1}, w_{j-1}, w_{j+1}$ ;
- 5831 • the prefixes, suffixes, and part-of-speech of these neighbor words.

5832 Each of these features can be conjoined with the dependency edge label  $r$ . Note that  
 5833 features in an arc-factored parser can refer to words other than  $w_i$  and  $w_j$ . The restriction  
 5834 is that the features consider only a single arc.

**Bilexical features** (e.g., *sushi* → *chopsticks*) are powerful but rare, so it is useful to augment them with coarse-grained alternatives, by “backing off” to the part-of-speech or affix. For example, the following features are created by backing off to part-of-speech tags in an unlabeled dependency parser:

$$\begin{aligned} f(3 \rightarrow 5, \text{we eat sushi with chopsticks}) = & \langle \text{sushi} \rightarrow \text{chopsticks}, \\ & \text{sushi} \rightarrow \text{NNS}, \\ & \text{NN} \rightarrow \text{chopsticks}, \\ & \text{NNS} \rightarrow \text{NN} \rangle. \end{aligned}$$

5835 Regularized discriminative learning algorithms can then trade off between features at  
 5836 varying levels of detail. McDonald et al. (2005) take this approach as far as *tetralexical*  
 5837 features (e.g.,  $(w_i, w_{i+1}, w_{j-1}, w_j)$ ). Such features help to avoid choosing arcs that are un-  
 5838 likely due to the intervening words: for example, there is unlikely to be an edge between  
 5839 two nouns if the intervening span contains a verb. A large list of first and second-order  
 5840 features is provided by Bohnet (2010), who uses a hashing function to store these features  
 5841 efficiently.

#### 5842 11.2.2.2 Neural arc scores

Given vector representations  $\mathbf{x}_i$  for each word  $w_i$  in the input, a set of arc scores can be computed from a feedforward neural network:

$$\psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \text{FeedForward}([\mathbf{x}_i; \mathbf{x}_j]; \boldsymbol{\theta}_r), \quad [11.7]$$

where unique weights  $\boldsymbol{\theta}_r$  are available for each arc type (Pei et al., 2015; Kiperwasser and Goldberg, 2016). Kiperwasser and Goldberg (2016) use a feedforward network with a single hidden layer,

$$\mathbf{z} = g(\boldsymbol{\Theta}_r[\mathbf{x}_i; \mathbf{x}_j] + \mathbf{b}_r^{(z)}) \quad [11.8]$$

$$\psi(i \xrightarrow{r} j) = \boldsymbol{\beta}_r \mathbf{z} + \mathbf{b}_r^{(y)}, \quad [11.9]$$

5843 where  $\boldsymbol{\Theta}_r$  is a matrix,  $\boldsymbol{\beta}_r$  is a vector, each  $\mathbf{b}_r$  is a scalar, and the function  $g$  is an elementwise  
 5844 tanh activation function.

5845 The vector  $\mathbf{x}_i$  can be set equal to the word embedding, which may be pre-trained or  
 5846 learned by backpropagation (Pei et al., 2015). Alternatively, contextual information can  
 5847 be incorporated by applying a bidirectional recurrent neural network across the input, as  
 5848 described in § 7.6. The RNN hidden states at each word can be used as inputs to the arc  
 5849 scoring function (Kiperwasser and Goldberg, 2016).

5850 **11.2.2.3 Probabilistic arc scores**

If each arc score is equal to the log probability  $\log p(w_j, r \mid w_i)$ , then the sum of scores gives the log probability of the sentence and arc labels, by the chain rule. For example, consider the unlabeled parse of *we eat sushi with rice*,

$$\mathbf{y} = \{(ROOT, 2), (2, 1), (2, 3), (3, 5), (5, 4)\} \quad [11.10]$$

$$\log p(\mathbf{w} \mid \mathbf{y}) = \sum_{(i \rightarrow j) \in \mathbf{y}} \log p(w_j \mid w_i) \quad [11.11]$$

$$\begin{aligned} &= \log p(eat \mid ROOT) + \log p(we \mid eat) + \log p(sushi \mid eat) \\ &\quad + \log p(rice \mid sushi) + \log p(with \mid rice). \end{aligned} \quad [11.12]$$

5851 Probabilistic generative models are used in combination with expectation-maximization  
5852 (chapter 5) for unsupervised dependency parsing (Klein and Manning, 2004).

5853 **11.2.3 Learning**

Having formulated graph-based dependency parsing as a structure prediction problem, we can apply similar learning algorithms to those used in sequence labeling. Given a loss function  $\ell(\boldsymbol{\theta}; \mathbf{w}^{(i)}, \mathbf{y}^{(i)})$ , we can compute gradient-based updates to the parameters. For a model with feature-based arc scores and a perceptron loss, we obtain the usual structured perceptron update,

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \mathbf{y}') \quad [11.13]$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \mathbf{f}(\mathbf{w}, \mathbf{y}) - \mathbf{f}(\mathbf{w}, \hat{\mathbf{y}}) \quad [11.14]$$

5854 In this case, the argmax requires a maximization over all dependency trees for the sen-  
5855 tence, which can be computed using the algorithms described in § 11.2.1. We can apply  
5856 all the usual tricks from § 2.2: weight averaging, a large margin objective, and regular-  
5857 ization. McDonald et al. (2005) were the first to treat dependency parsing as a structure  
5858 prediction problem, using MIRA, an online margin-based learning algorithm. Neural arc  
5859 scores can be learned in the same way, backpropagating from a margin loss to updates on  
5860 the feedforward network that computes the score for each edge.

A conditional random field for arc-factored dependency parsing is built on the probability model,

$$p(\mathbf{y} \mid \mathbf{w}) = \frac{\exp \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})}{\sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp \sum_{i \xrightarrow{r} j \in \mathbf{y}'} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})} \quad [11.15]$$

5861 Such a model is trained to minimize the negative log conditional-likelihood. Just as in  
5862 CRF sequence models (§ 7.5.3) and the logistic regression classifier (§ 2.4), the gradients

5863 involve marginal probabilities  $p(i \xrightarrow{r} j \mid \mathbf{w}; \theta)$ , which in this case are probabilities over  
 5864 individual dependencies. In arc-factored models, these probabilities can be computed  
 5865 in polynomial time. For projective dependency trees, the marginal probabilities can be  
 5866 computed in cubic time, using a variant of the inside-outside algorithm (Lari and Young,  
 5867 1990). For non-projective dependency parsing, marginals can also be computed in cubic  
 5868 time, using the **matrix-tree theorem** (Koo et al., 2007; McDonald et al., 2007; Smith and  
 5869 Smith, 2007). Details of these methods are described by Kübler et al. (2009).

### 5870 11.3 Transition-based dependency parsing

5871 Graph-based dependency parsing offers exact inference, meaning that it is possible to re-  
 5872 cover the best-scoring parse for any given model. But this comes at a price: the scoring  
 5873 function is required to decompose into local parts — in the case of non-projective parsing,  
 5874 these parts are restricted to individual arcs. These limitations are felt more keenly in de-  
 5875 pendency parsing than in sequence labeling, because second-order dependency features  
 5876 are critical to correctly identify some types of attachments. For example, prepositional  
 5877 phrase attachment depends on the attachment point, the object of the preposition, and  
 5878 the preposition itself; arc-factored scores cannot account for all three of these features si-  
 5879 multaneously. Graph-based dependency parsing may also be criticized on the basis of  
 5880 intuitions about human language processing: people read and listen to sentences *sequen-*  
 5881 *tially*, incrementally building mental models of the sentence structure and meaning before  
 5882 getting to the end (Jurafsky, 1996). This seems hard to reconcile with graph-based algo-  
 5883 rithms, which perform bottom-up operations on the entire sentence, requiring the parser  
 5884 to keep every word in memory. Finally, from a practical perspective, graph-based depen-  
 5885 dency parsing is relatively slow, running in cubic time in the length of the input.

5886 Transition-based algorithms address all three of these objections. They work by mov-  
 5887 ing through the sentence sequentially, while performing actions that incrementally up-  
 5888 date a stored representation of what has been read thus far. As with the shift-reduce  
 5889 parser from § 10.6.2, this representation consists of a stack, onto which parsing substruc-  
 5890 tures can be pushed and popped. In shift-reduce, these substructures were constituents;  
 5891 in the transition systems that follow, they will be projective dependency trees over partial  
 5892 spans of the input.<sup>4</sup> Parsing is complete when the input is consumed and there is only  
 5893 a single structure on the stack. The sequence of actions that led to the parse is known as  
 5894 the **derivation**. One problem with transition-based systems is that there may be multiple  
 5895 derivations for a single parse structure — a phenomenon known as **spurious ambiguity**.

---

<sup>4</sup>Transition systems also exist for non-projective dependency parsing (e.g., Nivre, 2008).

5896 **11.3.1 Transition systems for dependency parsing**

5897 A **transition system** consists of a representation for describing configurations of the parser,  
 5898 and a set of transition actions, which manipulate the configuration. There are two main  
 5899 transition systems for dependency parsing: **arc-standard**, which is closely related to shift-  
 5900 reduce, and **arc-eager**, which adds an additional action that can simplify derivations (Ab-  
 5901 ney and Johnson, 1991). In both cases, transitions are between **configurations** that are  
 5902 represented as triples,  $C = (\sigma, \beta, A)$ , where  $\sigma$  is the stack,  $\beta$  is the input buffer, and  $A$  is  
 5903 the list of arcs that have been created (Nivre, 2008). In the initial configuration,

$$C_{\text{initial}} = ([\text{ROOT}], \mathbf{w}, \emptyset), \quad [11.16]$$

5904 indicating that the stack contains only the special node ROOT, the entire input is on the  
 5905 buffer, and the set of arcs is empty. An accepting configuration is,

$$C_{\text{accept}} = ([\text{ROOT}], \emptyset, A), \quad [11.17]$$

5906 where the stack contains only ROOT, the buffer is empty, and the arcs  $A$  define a spanning  
 5907 tree over the input. The arc-standard and arc-eager systems define a set of transitions  
 5908 between configurations, which are capable of transforming an initial configuration into  
 5909 an accepting configuration. In both of these systems, the number of actions required to  
 5910 parse an input grows linearly in the length of the input, making transition-based parsing  
 5911 considerably more efficient than graph-based methods.

5912 **11.3.1.1 Arc-standard**

5913 The **arc-standard** transition system is closely related to shift-reduce, and to the LR algo-  
 5914 rithm that is used to parse programming languages (Aho et al., 2006). It includes the  
 5915 following classes of actions:

- 5916 • SHIFT: move the first item from the input buffer on to the top of the stack,

$$(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A), \quad [11.18]$$

5917 where we write  $i|\beta$  to indicate that  $i$  is the leftmost item in the input buffer, and  $\sigma|i$   
 5918 to indicate the result of pushing  $i$  on to stack  $\sigma$ .

- 5919 • ARC-LEFT: create a new left-facing arc of type  $r$  between the item on the top of the  
 5920 stack and the first item in the input buffer. The head of this arc is  $j$ , which remains  
 5921 at the front of the input buffer. The arc  $j \xrightarrow{r} i$  is added to  $A$ . Formally,

$$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \oplus j \xrightarrow{r} i), \quad [11.19]$$

5922 where  $r$  is the label of the dependency arc, and  $\oplus$  concatenates the new arc  $j \xrightarrow{r} i$  to  
 5923 the list  $A$ .

$\sigma$	$\beta$	action	arc added to $\mathcal{A}$
1. [ROOT]	<i>they like bagels with lox</i>	SHIFT	
2. [ROOT, <i>they</i> ]	<i>like bagels with lox</i>	ARC-LEFT	( <i>they</i> $\leftarrow$ <i>like</i> )
3. [ROOT]	<i>like bagels with lox</i>	SHIFT	
4. [ROOT, <i>like</i> ]	<i>bagels with lox</i>	SHIFT	
5. [ROOT, <i>like</i> , <i>bagels</i> ]	<i>with lox</i>	SHIFT	
6. [ROOT, <i>like</i> , <i>bagels</i> , <i>with</i> ]	<i>lox</i>	ARC-LEFT	( <i>with</i> $\leftarrow$ <i>lox</i> )
7. [ROOT, <i>like</i> , <i>bagels</i> ]	<i>lox</i>	ARC-RIGHT	( <i>bagels</i> $\rightarrow$ <i>lox</i> )
8. [ROOT, <i>like</i> ]	<i>bagels</i>	ARC-RIGHT	( <i>like</i> $\rightarrow$ <i>bagels</i> )
9. [ROOT]	<i>like</i>	ARC-RIGHT	(ROOT $\rightarrow$ <i>like</i> )
10. [ROOT]	$\emptyset$	DONE	

Table 11.2: Arc-standard derivation of the unlabeled dependency parse for the input *they like bagels with lox*.

- 5924 • ARC-RIGHT: creates a new right-facing arc of type  $r$  between the item on the top of  
 5925 the stack and the first item in the input buffer. The head of this arc is  $i$ , which is  
 5926 “popped” from the stack and pushed to the front of the input buffer. The arc  $i \xrightarrow{r} j$   
 5927 is added to  $A$ . Formally,

$$(\sigma | i, j | \beta, A) \Rightarrow (\sigma, i | \beta, A \oplus i \xrightarrow{r} j), \quad [11.20]$$

5928 where again  $r$  is the label of the dependency arc.

5929 Each action has preconditions. The SHIFT action can be performed only when the buffer  
 5930 has at least one element. The ARC-LEFT action cannot be performed when the root node  
 5931 ROOT is on top of the stack, since this node must be the root of the entire tree. The ARC-  
 5932 LEFT and ARC-RIGHT remove the modifier words from the stack (in the case of ARC-LEFT)  
 5933 and from the buffer (in the case of ARC-RIGHT), so it is impossible for any word to have  
 5934 more than one parent. Furthermore, the end state can only be reached when every word is  
 5935 removed from the buffer and stack, so the set of arcs is guaranteed to constitute a spanning  
 5936 tree. An example arc-standard derivation is shown in Table 11.2.

### 5937 11.3.1.2 Arc-eager dependency parsing

5938 In the arc-standard transition system, a word is completely removed from the parse once  
 5939 it has been made the modifier in a dependency arc. At this time, any dependents of  
 5940 this word must have already been identified. Right-branching structures are common in  
 5941 English (and many other languages), with words often modified by units such as prepo-  
 5942 sitional phrases to their right. In the arc-standard system, this means that we must first  
 5943 shift all the units of the input onto the stack, and then work backwards, creating a series of

5944 arcs, as occurs in Table 11.2. Note that the decision to shift *bagels* onto the stack guarantees  
 5945 that the prepositional phrase *with lox* will attach to the noun phrase, and that this decision  
 5946 must be made before the prepositional phrase is itself parsed. This has been argued to be  
 5947 cognitively implausible (Abney and Johnson, 1991); from a computational perspective, it  
 5948 means that a parser may need to look several steps ahead to make the correct decision.

5949 **Arc-eager dependency parsing** changes the ARC-RIGHT action so that right depen-  
 5950 dents can be attached before all of their dependents have been found. Rather than re-  
 5951 moving the modifier from both the buffer and stack, the ARC-RIGHT action pushes the  
 5952 modifier on to the stack, on top of the head. Because the stack can now contain elements  
 5953 that already have parents in the partial dependency graph, two additional changes are  
 5954 necessary:

- 5955 • A precondition is required to ensure that the ARC-LEFT action cannot be applied  
 5956 when the top element on the stack already has a parent in  $A$ .
- 5957 • A new REDUCE action is introduced, which can remove elements from the stack if  
 5958 they already have a parent in  $A$ :

$$(\sigma|i, \beta, A) \Rightarrow (\sigma, \beta, A). \quad [11.21]$$

5959 As a result of these changes, it is now possible to create the arc *like*  $\rightarrow$  *bagels* before parsing  
 5960 the prepositional phrase *with lox*. Furthermore, this action does not imply a decision about  
 5961 whether the prepositional phrase will attach to the noun or verb. Noun attachment is  
 5962 chosen in the parse in Table 11.3, but verb attachment could be achieved by applying the  
 5963 REDUCE action at step 5 or 7.

### 5964 11.3.1.3 Projectivity

5965 The arc-standard and arc-eager transition systems are guaranteed to produce projective  
 5966 dependency trees, because all arcs are between the word at the top of the stack and the  
 5967 left-most edge of the buffer (Nivre, 2008). Non-projective transition systems can be con-  
 5968 structed by adding actions that create arcs with words that are second or third in the  
 5969 stack (Attardi, 2006), or by adopting an alternative configuration structure, which main-  
 5970 tains a list of all words that do not yet have heads (Covington, 2001). In **pseudo-projective**  
 5971 **dependency parsing**, a projective dependency parse is generated first, and then a set of  
 5972 graph transformation techniques are applied, producing non-projective edges (Nivre and  
 5973 Nilsson, 2005).

### 5974 11.3.1.4 Beam search

5975 In “greedy” transition-based parsing, the parser tries to make the best decision at each  
 5976 configuration. This can lead to search errors, when an early decision locks the parser into

$\sigma$	$\beta$	action	arc added to $\mathcal{A}$
1. [ROOT]	<i>they like bagels with lox</i>	SHIFT	
2. [ROOT, <i>they</i> ]	<i>like bagels with lox</i>	ARC-LEFT	( <i>they</i> $\leftarrow$ <i>like</i> )
3. [ROOT]	<i>like bagels with lox</i>	ARC-RIGHT	(ROOT $\rightarrow$ <i>like</i> )
4. [ROOT, <i>like</i> ]	<i>bagels with lox</i>	ARC-RIGHT	( <i>like</i> $\rightarrow$ <i>bagels</i> )
5. [ROOT, <i>like</i> , <i>bagels</i> ]	<i>with lox</i>	SHIFT	
6. [ROOT, <i>like</i> , <i>bagels</i> , <i>with</i> ]	<i>lox</i>	ARC-LEFT	( <i>with</i> $\leftarrow$ <i>lox</i> )
7. [ROOT, <i>like</i> , <i>bagels</i> ]	<i>lox</i>	ARC-RIGHT	( <i>bagels</i> $\rightarrow$ <i>lox</i> )
8. [ROOT, <i>like</i> , <i>bagels</i> , <i>lox</i> ]	$\emptyset$	REDUCE	
9. [ROOT, <i>like</i> , <i>bagels</i> ]	$\emptyset$	REDUCE	
10. [ROOT, <i>like</i> ]	$\emptyset$	REDUCE	
11. [ROOT]	$\emptyset$	DONE	

Table 11.3: Arc-eager derivation of the unlabeled dependency parse for the input *they like bagels with lox*.

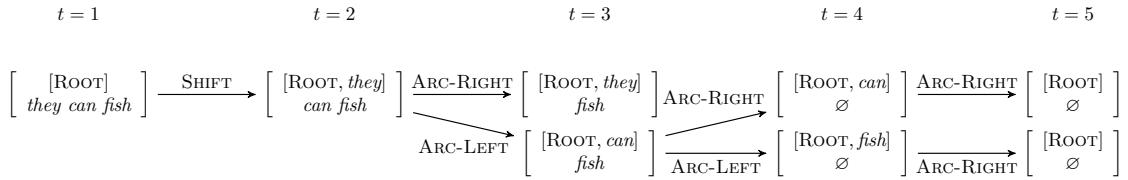


Figure 11.7: Beam search for unlabeled dependency parsing, with beam size  $K = 2$ . The arc lists for each configuration are not shown, but can be computed from the transitions.

5977 a poor derivation. For example, in Table 11.2, if ARC-RIGHT were chosen at step 4, then  
 5978 the parser would later be forced to attach the prepositional phrase *with lox* to the verb  
 5979 *likes*. Note that the *likes*  $\rightarrow$  *bagels* arc is indeed part of the correct dependency parse, but  
 5980 the arc-standard transition system requires it to be created later in the derivation.

Beam search addresses this issue by maintaining a set of hypothetical derivations, called a beam. At step  $t$  of the derivation, there is a set of  $k$  hypotheses, each of which is a tuple of a score and a sequence of actions,

$$h_t^{(k)} = (s_t^{(k)}, A_t^{(k)}) \quad [11.22]$$

5981 Each hypothesis is then “expanded” by considering the set of all valid actions from the  
 5982 current configuration  $c_t^{(k)}$ , written  $\mathcal{A}(c_t^{(k)})$ . This yields a large set of new hypotheses. For  
 5983 each action  $a \mathcal{A}(c_t^{(k)})$ , we score the new hypothesis  $A_t^{(k)} \oplus a$ . The top  $k$  hypotheses by  
 5984 this scoring metric are kept, and parsing proceeds to the next step (Zhang and Clark,

5985 2008). Note that beam search requires a scoring function for action *sequences*, rather than  
 5986 individual actions. This issue will be revisited in the next section.

5987 An example of beam search is shown in Figure 11.7, with a beam size of  $K = 2$ . For the  
 5988 first transition, the only valid action is SHIFT, so there is only one possible configuration  
 5989 at  $t = 2$ . From this configuration, there are three possible actions. The top two are ARC-  
 5990 RIGHT and ARC-LEFT, and so the resulting hypotheses from these actions are on the beam  
 5991 at  $t = 3$ . From these configurations, there are three possible actions each, but the best  
 5992 two are expansions of the bottom hypothesis at  $t = 3$ . Parsing continues until  $t = 5$ , at  
 5993 which point both hypotheses reach an accepting state. The best-scoring hypothesis is then  
 5994 selected as the parse.

### 5995 11.3.2 Scoring functions for transition-based parsers

Transition-based parsing requires selecting a series of actions. In greedy transition-based  
 parsing, this can be done by training a classifier,

$$\hat{a} = \underset{a \in \mathcal{A}(c)}{\operatorname{argmax}} \Psi(a, c, \mathbf{w}; \boldsymbol{\theta}), \quad [11.23]$$

5996 where  $\mathcal{A}(c)$  is the set of admissible actions in the current configuration  $c$ ,  $\mathbf{w}$  is the input,  
 5997 and  $\Psi$  is a scoring function with parameters  $\boldsymbol{\theta}$  (Yamada and Matsumoto, 2003).

5998 A feature-based score can be computed,  $\Psi(a, c, \mathbf{w}) = \boldsymbol{\theta} \cdot \mathbf{f}(a, c, \mathbf{w})$ , using features that  
 5999 may consider any aspect of the current configuration and input sequence. Typical features  
 6000 for transition-based dependency parsing include: the word and part-of-speech of the top  
 6001 element on the stack; the word and part-of-speech of the first, second, and third elements  
 6002 on the input buffer; pairs and triples of words and parts-of-speech from the top of the  
 6003 stack and the front of the buffer; the distance (in tokens) between the element on the top  
 6004 of the stack and the element in the front of the input buffer; the number of modifiers of  
 6005 each of these elements; and higher-order dependency features as described above in the  
 6006 section on graph-based dependency parsing (see, e.g., Zhang and Nivre, 2011).

6007 Parse actions can also be scored by neural networks. For example, Chen and Manning  
 6008 (2014) build a feedforward network in which the input layer consists of the concatenation  
 6009 of embeddings of several words and tags:

- 6010 • the top three words on the stack, and the first three words on the buffer;
- 6011 • the first and second leftmost and rightmost children (dependents) of the top two  
   6012 words on the stack;
- 6013 • the leftmost and right most grandchildren of the top two words on the stack;
- 6014 • embeddings of the part-of-speech tags of these words.

Let us call this base layer  $\mathbf{x}(c, \mathbf{w})$ , defined as,

$$c = (\sigma, \beta, A)$$

$$\mathbf{x}(c, \mathbf{w}) = [\mathbf{v}_{w_{\sigma_1}}, \mathbf{v}_{t_{\sigma_1}} \mathbf{v}_{w_{\sigma_2}}, \mathbf{v}_{t_{\sigma_2}}, \mathbf{v}_{w_{\sigma_3}}, \mathbf{v}_{t_{\sigma_3}}, \mathbf{v}_{w_{\beta_1}}, \mathbf{v}_{t_{\beta_1}}, \mathbf{v}_{w_{\beta_2}}, \mathbf{v}_{t_{\beta_2}}, \dots],$$

where  $\mathbf{v}_{w_{\sigma_1}}$  is the embedding of the first word on the stack,  $\mathbf{v}_{t_{\beta_2}}$  is the embedding of the part-of-speech tag of the second word on the buffer, and so on. Given this base encoding of the parser state, the score for the set of possible actions is computed through a feedforward network,

$$\mathbf{z} = g(\Theta^{(x \rightarrow z)} \mathbf{x}(c, \mathbf{w})) \quad [11.24]$$

$$\psi(a, c, \mathbf{w}; \boldsymbol{\theta}) = \Theta_a^{(z \rightarrow y)} \mathbf{z}, \quad [11.25]$$

6015 where the vector  $\mathbf{z}$  plays the same role as the features  $\mathbf{f}(a, c, \mathbf{w})$ , but is a learned representation.  
 6016 Chen and Manning (2014) use a cubic elementwise activation function,  $g(x) = x^3$ ,  
 6017 so that the hidden layer models products across all triples of input features. The learning  
 6018 algorithm updates the embeddings as well as the parameters of the feedforward network.

### 6019 11.3.3 Learning to parse

6020 Transition-based dependency parsing suffers from a mismatch between the supervision,  
 6021 which comes in the form of dependency trees, and the classifier's prediction space, which  
 6022 is a set of parsing actions. One solution is to create new training data by converting parse  
 6023 trees into action sequences; another is to derive supervision directly from the parser's  
 6024 performance.

#### 6025 11.3.3.1 Oracle-based training

6026 A transition system can be viewed as a function from action sequences (also called **derivations**)  
 6027 to parse trees. The inverse of this function is a mapping from parse trees to derivations,  
 6028 which is called an **oracle**. For the arc-standard and arc-eager parsing system, an  
 6029 oracle can be computed in linear time in the length of the derivation (Kübler et al., 2009,  
 6030 page 32). Both the arc-standard and arc-eager transition systems suffer from **spurious**  
 6031 **ambiguity**: there exist dependency parses for which multiple derivations are possible,  
 6032 such as  $1 \leftarrow 2 \rightarrow 3$ . The oracle must choose between these different derivations. For exam-  
 6033 ple, the algorithm described by Kübler et al. (2009) would first create the left arc ( $1 \leftarrow 2$ ),  
 6034 and then create the right arc,  $(1 \leftarrow 2) \rightarrow 3$ ; another oracle might begin by shifting twice,  
 6035 resulting in the derivation  $1 \leftarrow (2 \rightarrow 3)$ .

Given such an oracle, a dependency treebank can be converted into a set of oracle action sequences  $\{A^{(i)}\}_{i=1}^N$ . The parser can be trained by stepping through the oracle action sequences, and optimizing on an classification-based objective that rewards selecting the

oracle action. For transition-based dependency parsing, maximum conditional likelihood is a typical choice (Chen and Manning, 2014; Dyer et al., 2015):

$$p(a | c, \mathbf{w}) = \frac{\exp \Psi(a, c, \mathbf{w}; \boldsymbol{\theta})}{\sum_{a' \in \mathcal{A}(c)} \exp \Psi(a', c, \mathbf{w}; \boldsymbol{\theta})} \quad [11.26]$$

$$\hat{\boldsymbol{\theta}} = \operatorname{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^N \sum_{t=1}^{|A^{(i)}|} \log p(a_t^{(i)} | c_t^{(i)}, \mathbf{w}), \quad [11.27]$$

6036 where  $|A^{(i)}|$  is the length of the action sequence  $A^{(i)}$ .

6037 Recall that beam search requires a scoring function for action sequences. Such a score  
 6038 can be obtained by adding the log-likelihoods (or hinge losses) across all actions in the  
 6039 sequence (Chen and Manning, 2014).

#### 6040 11.3.3.2 Global objectives

6041 The objective in Equation 11.27 is **locally-normalized**: it is the product of normalized  
 6042 probabilities over individual actions. A similar characterization could be made of non-  
 6043 probabilistic algorithms in which hinge-loss objectives are summed over individual ac-  
 6044 tions. In either case, training on individual actions can be sub-optimal with respect to  
 6045 global performance, due to the **label bias problem** (Lafferty et al., 2001; Andor et al.,  
 6046 2016).

6047 As a stylized example, suppose that a given configuration appears 100 times in the  
 6048 training data, with action  $a_1$  as the oracle action in 51 cases, and  $a_2$  as the oracle action in  
 6049 the other 49 cases. However, in cases where  $a_2$  is correct, choosing  $a_1$  results in a cascade  
 6050 of subsequent errors, while in cases where  $a_1$  is correct, choosing  $a_2$  results in only a single  
 6051 error. A classifier that is trained on a local objective function will learn to always choose  
 6052  $a_1$ , but choosing  $a_2$  would minimize the overall number of errors.

6053 This observation motivates a global objective, such as the globally-normalized condi-  
 6054 tional likelihood,

$$p(A^{(i)} | \mathbf{w}; \boldsymbol{\theta}) = \frac{\exp \sum_{t=1}^{|A^{(i)}|} \Psi(a_t^{(i)}, c_t^{(i)}, \mathbf{w})}{\sum_{A' \in \mathbb{A}(\mathbf{w})} \exp \sum_{t=1}^{|A'|} \Psi(a'_t, c'_t, \mathbf{w})}, \quad [11.28]$$

where the denominator sums over the set of all possible action sequences,  $\mathbb{A}(\mathbf{w})$ .<sup>5</sup> In the conditional random field model for sequence labeling (§ 7.5.3), it was possible to compute

---

<sup>5</sup>Andor et al. (2016) prove that the set of globally-normalized conditional distributions is a strict superset of the set of locally-normalized conditional distributions, and that globally-normalized conditional models are therefore strictly more expressive.

this sum explicitly, using dynamic programming. In transition-based parsing, this is not possible. However, the sum can be approximated using beam search,

$$\sum_{A' \in \mathbb{A}(\mathbf{w})} \exp \sum_{t=1}^{|A'|} \Psi(a'_t, c'_t, \mathbf{w}) \approx \sum_{k=1}^K \exp \sum_{t=1}^{|A^{(k)}|} \Psi(a_t^{(k)}, c_t^{(k)}, \mathbf{w}), \quad [11.29]$$

where  $A^{(k)}$  is an action sequence on a beam of size  $K$ . This gives rise to the following loss function,

$$L(\boldsymbol{\theta}) = - \sum_{t=1}^{|A^{(i)}|} \Psi(a_t^{(i)}, c_t^{(i)}, \mathbf{w}) + \log \sum_{k=1}^K \exp \sum_{t=1}^{|A^{(k)}|} \Psi(a_t^{(k)}, c_t^{(k)}, \mathbf{w}). \quad [11.30]$$

6055 The derivatives of this loss involve expectations with respect to a probability distribution  
6056 over action sequences on the beam.

### 6057 11.3.3.3 \*Early update and the incremental perceptron

6058 When learning in the context of beam search, the goal is to learn a decision function so that  
6059 the gold dependency parse is always reachable from at least one of the partial derivations  
6060 on the beam. (The combination of a transition system (such as beam search) and a scoring  
6061 function for actions is known as a **policy**.) To achieve this, we can make an **early update**  
6062 as soon as the oracle action sequence “falls off” the beam, even before a complete analysis  
6063 is available (Collins and Roark, 2004; Daumé III and Marcu, 2005). The loss can be based  
6064 on the best-scoring hypothesis on the beam, or the sum of all hypotheses (Huang et al.,  
6065 2012).

6066 For example, consider the beam search in Figure 11.7. In the correct parse, *fish* is the  
6067 head of dependency arcs to both of the other two words. In the arc-standard system,  
6068 this can be achieved only by using SHIFT for the first two actions. At  $t = 3$ , the oracle  
6069 action sequence has fallen off the beam. The parser should therefore stop, and update the  
6070 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  
6071 oracle sequence, and  $\{A_{1:3}^{(k)}\}$  is the beam.

6072 This integration of incremental search and learning was first developed in the **incremental**  
6073 **perceptron** (Collins and Roark, 2004). This method updates the parameters with  
6074 respect to a hinge loss, which compares the top-scoring hypothesis and the gold action  
6075 sequence, up to the current point  $t$ . Several improvements to this basic protocol are pos-  
6076 sible:

- 6077 • As noted earlier, the gold dependency parse can be derived by multiple action se-  
6078 quences. Rather than checking for the presence of a single oracle action sequence on  
6079 the beam, we can check if the gold dependency parse is *reachable* from the current  
6080 beam, using a **dynamic oracle** (Goldberg and Nivre, 2012).

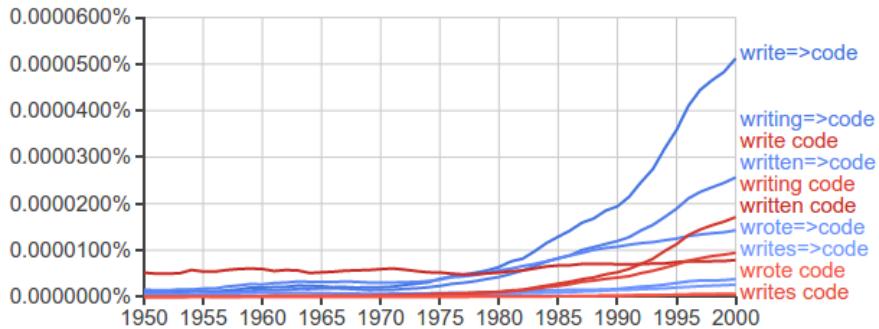


Figure 11.8: Google n-grams results for the bigram *write code* and the dependency arc *write => code* (and their morphological variants)

- By maximizing the score of the gold action sequence, we are training a decision function to find the correct action given the gold context. But in reality, the parser will make errors, and the parser is not trained to find the best action given a context that may not itself be optimal. This issue is addressed by various generalizations of incremental perceptron, known as **learning to search** (Daumé III et al., 2009). Some of these methods are discussed in chapter 15.

## 11.4 Applications

Dependency parsing is used in many real-world applications: any time you want to know about pairs of words which might not be adjacent, you can use dependency arcs instead of regular expression search patterns. For example, you may want to match strings like *delicious pastries*, *delicious French pastries*, and *the pastries are delicious*.

It is possible to search the Google *n*-gramscorpus by dependency edges, finding the trend in how often a dependency edge appears over time. For example, we might be interested in knowing when people started talking about *writing code*, but we also want *write some code*, *write good code*, *write all the code*, etc. The result of a search on the dependency edge *write → code* is shown in Figure 11.8. This capability has been applied to research in digital humanities, such as the analysis of gender in Shakespeare Muralidharan and Hearst (2013).

A classic application of dependency parsing is **relation extraction**, which is described

in chapter 17. The goal of relation extraction is to identify entity pairs, such as

(MELVILLE, MOBY-DICK)  
 (TOLSTOY, WAR AND PEACE)  
 (MARQUÉZ, 100 YEARS OF SOLITUDE)  
 (SHAKESPEARE, A MIDSUMMER NIGHT'S DREAM),

6099 which stand in some relation to each other (in this case, the relation is authorship). Such  
 6100 entity pairs are often referenced via consistent chains of dependency relations. Therefore,  
 6101 dependency paths are often a useful feature in supervised systems which learn to detect  
 6102 new instances of a relation, based on labeled examples of other instances of the same  
 6103 relation type (Culotta and Sorensen, 2004; Fundel et al., 2007; Mintz et al., 2009).

6104 Cui et al. (2005) show how dependency parsing can improve automated question an-  
 6105 swering. Suppose you receive the following query:

6106 (11.1) What percentage of the nation's cheese does Wisconsin produce?

6107 The corpus contains this sentence:

6108 (11.2) In Wisconsin, where farmers produce 28% of the nation's cheese, ...

6109 The location of *Wisconsin* in the surface form of this string makes it a poor match for the  
 6110 query. However, in the dependency graph, there is an edge from *produce* to *Wisconsin* in  
 6111 both the question and the potential answer, raising the likelihood that this span of text is  
 6112 relevant to the question.

6113 A final example comes from sentiment analysis. As discussed in chapter 4, the polarity  
 6114 of a sentence can be reversed by negation, e.g.

6115 (11.3) *There is no reason at all to believe the polluters will suddenly become reasonable.*

6116 By tracking the sentiment polarity through the dependency parse, we can better iden-  
 6117 tify the overall polarity of the sentence, determining when key sentiment words are re-  
 6118 versed (Wilson et al., 2005; Nakagawa et al., 2010).

## 6119 Additional resources

6120 More details on dependency grammar and parsing algorithms can be found in the manuscript  
 6121 by Kübler et al. (2009). For a comprehensive but whimsical overview of graph-based de-  
 6122 pendency parsing algorithms, see Eisner (1997). Jurafsky and Martin (2018) describe an  
 6123 **agenda-based** version of beam search, in which the beam contains hypotheses of varying  
 6124 lengths. New hypotheses are added to the beam only if their score is better than the worst

item currently on the beam. Another search algorithm for transition-based parsing is **easy-first**, which abandons the left-to-right traversal order, and adds the highest-scoring edges first, regardless of where they appear (Goldberg and Elhadad, 2010). Goldberg et al. (2013) note that although transition-based methods can be implemented in linear time in the length of the input, naïve implementations of beam search will require quadratic time, due to the cost of copying each hypothesis when it is expanded on the beam. This issue can be addressed by using a more efficient data structure for the stack.

## Exercises

1. The dependency structure  $1 \leftarrow 2 \rightarrow 3$ , with 2 as the root, can be obtained from more than one set of actions in arc-standard parsing. List both sets of actions that can obtain this parse.
2. Suppose you have a set of unlabeled arc scores  $\psi(i \rightarrow j)$ , where the score depends only on the identity of the two words. The scores include  $\psi(\text{ROOT} \rightarrow j)$ .
  - Assuming each word occurs only once in the sentence ( $(i \neq j) \Leftarrow (w_i \neq w_j)$ ), how would you construct a weighted lexicalized context-free grammar so that the score of *any* projective dependency tree is equal to the score of some equivalent derivation in the lexicalized context-free grammar?
  - Verify that your method works for a simple example like *they eat fish*.
  - How would you adapt your method to handle the case an individual word may appear multiple times in the sentence?
3. Provide the UD-style dependency parse for the sentence *Xi-Lan eats shoots and leaves*, assuming *leaves* is a verb. Provide arc-standard and arc-eager derivations for this dependency parse.



6148

## **Part III**

6149

# **Meaning**



## 6150 Chapter 12

# 6151 Logical semantics

6152 The previous few chapters have focused on building systems that reconstruct the **syntax**  
6153 of natural language — its structural organization — through tagging and parsing. But  
6154 some of the most exciting and promising potential applications of language technology  
6155 involve going beyond syntax to **semantics** — the underlying meaning of the text:

- 6156 • Answering questions, such as *where is the nearest coffeeshop?* or *what is the middle name*  
6157 *of the mother of the 44th President of the United States?*.
- 6158 • Building a robot that can follow natural language instructions to execute tasks.
- 6159 • Translating a sentence from one language into another, while preserving the under-  
6160 lying meaning.
- 6161 • Fact-checking an article by searching the web for contradictory evidence.
- 6162 • Logic-checking an argument by identifying contradictions, ambiguity, and unsup-  
6163 ported assertions.

6164 Semantic analysis involves converting natural language into a **meaning representa-**  
6165 **tion**. To be useful, a meaning representation must meet several criteria:

- 6166 • **c1**: it should be unambiguous: unlike natural language, there should be exactly one  
6167 meaning per statement;
- 6168 • **c2**: it should provide a way to link language to external knowledge, observations,  
6169 and actions;
- 6170 • **c3**: it should support computational **inference**, so that meanings can be combined  
6171 to derive additional knowledge;
- 6172 • **c4**: it should be expressive enough to cover the full range of things that people talk  
6173 about in natural language.

6174 Much more than this can be said about the question of how best to represent knowledge  
 6175 for computation (e.g., Sowa, 2000), but this chapter will focus on these four criteria.

## 6176 12.1 Meaning and denotation

6177 The first criterion for a meaning representation is that statements in the representation  
 6178 should be unambiguous — they should have only one possible interpretation. Natural  
 6179 language does not have this property: as we saw in chapter 10, sentences like *cats scratch*  
 6180 *people with claws* have multiple interpretations.

6181 But what does it mean for a statement to be unambiguous? Programming languages  
 6182 provide a useful example: the output of a program is completely specified by the rules of  
 6183 the language and the properties of the environment in which the program is run. For ex-  
 6184 ample, the python code  $5 + 3$  will have the output 8, as will the codes  $(4 * 4) - (3 * 3) + 1$   
 6185 and  $((8))$ . This output is known as the **denotation** of the program, and can be written  
 6186 as,

$$\llbracket 5+3 \rrbracket = \llbracket (4 * 4) - (3 * 3) + 1 \rrbracket = \llbracket ((8)) \rrbracket = 8. \quad [12.1]$$

6187 The denotations of these arithmetic expressions are determined by the meaning of the  
 6188 **constants** (e.g., 5, 3) and the **relations** (e.g.,  $+$ ,  $*$ ,  $(,)$ ). Now let's consider another snippet  
 6189 of python code, `double(4)`. The denotation of this code could be,  $\llbracket \text{double}(4) \rrbracket = 8$ , or  
 6190 it could be  $\llbracket \text{double}(4) \rrbracket = 44$  — it depends on the meaning of `double`. This meaning  
 6191 is defined in a **world model**  $\mathcal{M}$  as an infinite set of pairs. We write the denotation with  
 6192 respect to model  $\mathcal{M}$  as  $\llbracket \cdot \rrbracket_{\mathcal{M}}$ , e.g.,  $\llbracket \text{double} \rrbracket_{\mathcal{M}} = \{(0, 0), (1, 2), (2, 4), \dots\}$ . The world  
 6193 model would also define the (infinite) list of constants, e.g.,  $\{0, 1, 2, \dots\}$ . As long as the  
 6194 denotation of string  $\phi$  in model  $\mathcal{M}$  can be computed unambiguously, the language can be  
 6195 said to be unambiguous.

6196 This approach to meaning is known as **model-theoretic semantics**, and it addresses  
 6197 not only criterion *c1* (no ambiguity), but also *c2* (connecting language to external knowl-  
 6198 edge, observations, and actions). For example, we can connect a representation of the  
 6199 meaning of a statement like *the capital of Georgia* with a world model that includes knowl-  
 6200 edge base of geographical facts, obtaining the denotation `Atlanta`. We might populate  
 6201 a world model by applying an image analysis algorithm to Figure 12.1, and then use this  
 6202 world model to evaluate **propositions** like *a man is riding a moose*. Another desirable prop-  
 6203 erty of model-theoretic semantics is that when the facts change, the denotations change  
 6204 too: the meaning representation of *President of the USA* would have a different denotation  
 6205 in the model  $\mathcal{M}_{2014}$  as it would in  $\mathcal{M}_{2022}$ .



Figure 12.1: A (doctored) image, which could be the basis for a world model

## 6206 12.2 Logical representations of meaning

6207 Criterion *c3* requires that the meaning representation support inference — for example,  
 6208 automatically deducing new facts from known premises. While many representations  
 6209 have been proposed that meet these criteria, the most mature is the language of first-order  
 6210 logic.<sup>1</sup>

### 6211 12.2.1 Propositional logic

6212 The bare bones of logical meaning representation are Boolean operations on propositions:

6213 **Propositional symbols.** Greek symbols like  $\phi$  and  $\psi$  will be used to represent **proposi-**  
 6214 **tions**, which are statements that are either true or false. For example,  $\phi$  may corre-  
 6215 spond to the proposition, *bagels are delicious*.

6216 **Boolean operators.** We can build up more complex propositional formulas from Boolean  
 6217 operators. These include:

- 6218 • Negation  $\neg\phi$ , which is true if  $\phi$  is false.

---

<sup>1</sup>Alternatives include the “variable-free” representation used in semantic parsing of geographical queries (Zelle and Mooney, 1996) and robotic control (Ge and Mooney, 2005), and dependency-based compositional semantics (Liang et al., 2013).

- 6219 • Conjunction,  $\phi \wedge \psi$ , which is true if both  $\phi$  and  $\psi$  are true.
- 6220 • Disjunction,  $\phi \vee \psi$ , which is true if at least one of  $\phi$  and  $\psi$  is true
- 6221 • Implication,  $\phi \Rightarrow \psi$ , which is true unless  $\phi$  is true and  $\psi$  is false. Implication  
6222 has identical truth conditions to  $\neg\phi \vee \psi$ .
- 6223 • Equivalence,  $\phi \Leftrightarrow \psi$ , which is true if  $\phi$  and  $\psi$  are both true or both false. Equiv-  
6224 alence has identical truth conditions to  $(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$ .

6225 It is not strictly necessary to have all five Boolean operators: readers familiar with  
6226 Boolean logic will know that it is possible to construct all other operators from either the  
6227 NAND (not-and) or NOR (not-or) operators. Nonetheless, it is clearest to use all five  
6228 operators. From the truth conditions for these operators, it is possible to define a number  
6229 of “laws” for these Boolean operators, such as,

- 6230 • *Commutativity*:  $\phi \wedge \psi = \psi \wedge \phi$ ,  $\phi \vee \psi = \psi \vee \phi$
- 6231 • *Associativity*:  $\phi \wedge (\psi \wedge \chi) = (\phi \wedge \psi) \wedge \chi$ ,  $\phi \vee (\psi \vee \chi) = (\phi \vee \psi) \vee \chi$
- 6232 • *Complementation*:  $\phi \wedge \neg\phi = \perp$ ,  $\phi \vee \neg\phi = \top$ , where  $\top$  indicates a true proposition  
6233 and  $\perp$  indicates a false proposition.

These laws can be combined to derive further equivalences, which can support logical inferences. For example, suppose  $\phi = \text{The music is loud}$  and  $\psi = \text{Max can't sleep}$ . Then if we are given,

$$\begin{aligned} \phi \Rightarrow \psi & \quad \text{If the music is loud, Max can't sleep.} \\ \phi & \quad \text{The music is loud.} \end{aligned}$$

6234 we can derive  $\psi$  (*Max can't sleep*) by application of **modus ponens**, which is one of a  
6235 set of **inference rules** that can be derived from more basic laws and used to manipulate  
6236 propositional formulas. **Automated theorem provers** are capable of applying inference  
6237 rules to a set of premises to derive desired propositions (Loveland, 2016).

### 6238 12.2.2 First-order logic

6239 Propositional logic is so named because it treats propositions as its base units. However,  
6240 the criterion *c4* states that our meaning representation should be sufficiently expressive.  
6241 Now consider the sentence pair,

- 6242 (12.1) If anyone is making noise, then Max can't sleep.  
6243 Abigail is making noise.

6244 People are capable of making inferences from this sentence pair, but such inferences re-  
6245 quire formal tools that are beyond propositional logic. To understand the relationship

6246 between the statement *anyone is making noise* and the statement *Abigail is making noise*, our  
 6247 meaning representation requires the additional machinery of **first-order logic** (FOL).

6248 In FOL, logical propositions can be constructed from relationships between entities.  
 6249 Specifically, FOL extends propositional logic with the following classes of terms:

6250 **Constants.** These are elements that name individual entities in the model, such as MAX  
 6251 and ABIGAIL. The denotation of each constant in a model  $\mathcal{M}$  is an element in the  
 6252 model, e.g.,  $\llbracket \text{MAX} \rrbracket = m$  and  $\llbracket \text{ABIGAIL} \rrbracket = a$ .

6253 **Relations.** Relations can be thought of as sets of entities, or sets of tuples. For example,  
 6254 the relation CAN-SLEEP is defined as the set of entities who can sleep, and has the  
 6255 denotation  $\llbracket \text{CAN-SLEEP} \rrbracket = \{a, m, \dots\}$ . To test the truth value of the proposition  
 6256 CAN-SLEEP(MAX), we ask whether  $\llbracket \text{MAX} \rrbracket \in \llbracket \text{CAN-SLEEP} \rrbracket$ . Logical relations that are  
 6257 defined over sets of entities are sometimes called **properties**.

6258 Relations may also be ordered tuples of entities. For example  $\text{BROTHER}(\text{MAX}, \text{ABIGAIL})$   
 6259 expresses the proposition that MAX is the brother of ABIGAIL. The denotation of  
 6260 such relations is a set of tuples,  $\llbracket \text{BROTHER} \rrbracket = \{(m, a), (x, y), \dots\}$ . To test the  
 6261 truth value of the proposition  $\text{BROTHER}(\text{MAX}, \text{ABIGAIL})$ , we ask whether the tuple  
 6262  $(\llbracket \text{MAX} \rrbracket, \llbracket \text{ABIGAIL} \rrbracket)$  is 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,

$$(\text{BROTHER}(\text{MAX}, \text{ABIGAIL}) \vee \text{BROTHER}(\text{MAX}, \text{STEVE})) \Rightarrow \neg \text{CAN-SLEEP}(\text{MAX}).$$

6263 This fragment of first-order logic permits only statements about specific entities. To  
 6264 support inferences about statements like *If anyone is making noise, then Max can't sleep*,  
 6265 two more elements must be added to the meaning representation:

6266 **Variables.** Variables are mechanisms for referring to entities that are not locally specified.  
 6267 We can then write  $\text{CAN-SLEEP}(x)$  or  $\text{BROTHER}(x, \text{ABIGAIL})$ . In these cases,  $x$  is a **free  
 6268 variable**, meaning that we have not committed to any particular assignment.

6269 **Quantifiers.** Variables are bound by quantifiers. There are two quantifiers in first-order  
 6270 logic.<sup>2</sup>

- 6271 • The **existential quantifier**  $\exists$ , which indicates that there must be at least one en-  
 6272 tity to which the variable can bind. For example, the statement  $\exists x \text{MAKES-NOISE}(x)$   
 6273 indicates that there is at least one entity for which MAKES-NOISE is true.  
 6274 • The **universal quantifier**  $\forall$ , which indicates that the variable must be able to  
 6275 bind to any entity in the model. For example, the statement,

$$\text{MAKES-NOISE(ABIGAIL)} \Rightarrow (\forall x \neg \text{CAN-SLEEP}(x)) \quad [12.3]$$

6276 asserts that if Abigail makes noise, no one can sleep.

6277 The expressions  $\exists x$  and  $\forall x$  make  $x$  into a **bound variable**. A formula that contains  
 6278 no free variables is a **sentence**.

6279 **Functions.** Functions map from entities to entities, e.g.,  $\llbracket \text{CAPITAL-OF(GEORGIA)} \rrbracket = \llbracket \text{ATLANTA} \rrbracket$ .  
 6280 With functions, it is convenient to add an equality operator, supporting statements  
 6281 like,

$$\forall x \exists y \text{MOTHER-OF}(x) = \text{DAUGHTER-OF}(y). \quad [12.4]$$

6282 Note that MOTHER-OF is a functional analogue of the relation MOTHER, so that  
 6283  $\text{MOTHER-OF}(x) = y$  if  $\text{MOTHER}(x, y)$ . Any logical formula that uses functions can be  
 6284 rewritten using only relations and quantification. For example,

$$\text{MAKES-NOISE}(\text{MOTHER-OF(ABIGAIL)}) \quad [12.5]$$

6285 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:

$$\forall x \exists y \text{ SPEAKS}(x, y) \quad [12.6]$$

$$\exists y \forall x \text{ SPEAKS}(x, y). \quad [12.7]$$

6286 In the first case,  $y$  may refer to several different languages, while in the second case, there  
 6287 is a single  $y$  that is spoken by everyone.

---

<sup>2</sup>In first-order logic, it is possible to quantify only over entities. In **second-order logic**, it is possible to quantify over properties, supporting statements like *Butch has every property that a good boxer has* (example from Blackburn and Bos, 2005),

$$\forall P \forall x ((\text{GOOD-BOXER}(x) \Rightarrow P(x)) \Rightarrow P(\text{BUTCH})). \quad [12.2]$$

## 6288 12.2.2.1 Truth-conditional semantics

6289 One way to look at the meaning of an FOL sentence  $\phi$  is as a set of **truth conditions**,  
 6290 or models under which  $\phi$  is satisfied. But how to determine whether a sentence is true  
 6291 or false in a given model? We will approach this inductively, starting with a predicate  
 6292 applied to a tuple of constants. The truth of such a sentence depends on whether the  
 6293 tuple of denotations of the constants is in the denotation of the predicate. For example,  
 6294 CAPITAL(GEORGIA,ATLANTA) is true in model  $\mathcal{M}$  iff,

$$(\llbracket \text{GEORGIA} \rrbracket_{\mathcal{M}}, \llbracket \text{ATLANTA} \rrbracket_{\mathcal{M}}) \in \llbracket \text{CAPITAL} \rrbracket_{\mathcal{M}}. \quad [12.8]$$

6295 The Boolean operators  $\wedge, \vee, \dots$  provide ways to construct more complicated sentences,  
 6296 and the truth of such statements can be assessed based on the truth tables associated with  
 6297 these operators. The statement  $\exists x\phi$  is true if there is some assignment of the variable  $x$   
 6298 to an entity in the model such that  $\phi$  is true; the statement  $\forall x\phi$  is true if  $\phi$  is true under  
 6299 all possible assignments of  $x$ . More formally, we would say that  $\phi$  is **satisfied** under  $\mathcal{M}$ ,  
 6300 written as  $\mathcal{M} \models \phi$ .

6301 Truth conditional semantics allows us to define several other properties of sentences  
 6302 and pairs of sentences. Suppose that in every  $\mathcal{M}$  under which  $\phi$  is satisfied, another  
 6303 formula  $\psi$  is also satisfied; then  $\phi$  **entails**  $\psi$ , which is also written as  $\phi \models \psi$ . For example,

$$\text{CAPITAL(GEORGIA,ATLANTA)} \models \exists x \text{CAPITAL(GEORGIA, } x\text{)}. \quad [12.9]$$

6304 A statement that is satisfied under any model, such as  $\phi \vee \neg\phi$ , is **valid**, written  $\models (\phi \vee$   
 6305  $\neg\phi)$ . A statement that is not satisfied under any model, such as  $\phi \wedge \neg\phi$ , is **unsatisfiable**,  
 6306 or **inconsistent**. A **model checker** is a program that determines whether a sentence  $\phi$   
 6307 is satisfied in  $\mathcal{M}$ . A **model builder** is a program that constructs a model in which  $\phi$   
 6308 is satisfied. The problems of checking for consistency and validity in first-order logic  
 6309 are **undecidable**, meaning that there is no algorithm that can automatically determine  
 6310 whether an FOL formula is valid or inconsistent.

## 6311 12.2.2.2 Inference in first-order logic

6312 Our original goal was to support inferences that combine general statements *If anyone is*  
*making noise, then Max can't sleep* with specific statements like *Abigail is making noise*. We  
 6313 can now represent such statements in first-order logic, but how are we to perform the  
 6314 inference that *Max can't sleep*? One approach is to use “generalized” versions of proposi-  
 6315 tional inference rules like modus ponens, which can be applied to FOL formulas. By  
 6316 repeatedly applying such inference rules to a knowledge base of facts, it is possible to  
 6317 produce proofs of desired propositions. To find the right sequence of inferences to derive  
 6318 a desired theorem, classical artificial intelligence search algorithms like backward chain-  
 6319 ing can be applied. Such algorithms are implemented in interpreters for the `prolog` logic  
 6320 programming language (Pereira and Shieber, 2002).

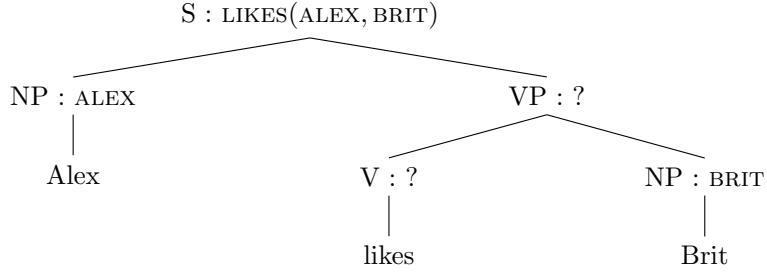


Figure 12.2: The principle of compositionality requires that we identify meanings for the constituents *likes* and *likes Brit* that will make it possible to compute the meaning for the entire sentence.

### 6322 12.3 Semantic parsing and the lambda calculus

6323 The previous section laid out a lot of formal machinery; the remainder of this chapter  
 6324 links these formalisms back to natural language. Given an English sentence like *Alex likes*  
 6325 *Brit*, how can we obtain the desired first-order logical representation,  $\text{LIKES}(\text{ALEX}, \text{BRIT})$ ?  
 6326 This is the task of **semantic parsing**. Just as a syntactic parser is a function from a natu-  
 6327 ral language sentence to a syntactic structure such as a phrase structure tree, a semantic  
 6328 parser is a function from natural language to logical formulas.

6329 As in syntactic analysis, semantic parsing is difficult because the space of inputs and  
 6330 outputs is very large, and their interaction is complex. Our best hope is that, like syntactic  
 6331 parsing, semantic parsing can somehow be decomposed into simpler sub-problems. This  
 6332 idea, usually attributed to the German philosopher Gottlob Frege, is called the **principle**  
 6333 **of compositionality**: the meaning of a complex expression is a function of the meanings of  
 6334 that expression's constituent parts. We will define these "constituent parts" as syntactic  
 6335 constituents: noun phrases and verb phrases. These constituents are combined using  
 6336 function application: if the syntactic parse contains the production  $x \rightarrow y z$ , then the  
 6337 semantics of  $x$ , written  $x.\text{sem}$ , will be computed as a function of the semantics of the  
 6338 constituents,  $y.\text{sem}$  and  $z.\text{sem}$ .<sup>3</sup> <sup>4</sup>

---

<sup>3</sup>§ 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.

<sup>4</sup>The 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).

6339 **12.3.1 The lambda calculus**

6340 Let's see how this works for a simple sentence like *Alex likes Brit*, whose syntactic structure  
 6341 is shown in Figure 12.2. Our goal is the formula, LIKES(ALEX,BRIT), and it is clear that the  
 6342 meaning of the constituents *Alex* and *Brit* should be ALEX and BRIT. That leaves two more  
 6343 constituents: the verb *likes*, and the verb phrase *likes Brit*. The meanings of these units  
 6344 must be defined in a way that makes it possible to recover the desired meaning for the  
 6345 entire sentence by function application. If the meanings of *Alex* and *Brit* are constants,  
 6346 then the meanings of *likes* and *likes Brit* must be functional expressions, which can be  
 6347 applied to their siblings to produce the desired analyses.

6348 Modeling these partial analyses requires extending the first-order logic meaning rep-  
 6349 resentation. We do this by adding **lambda expressions**, which are descriptions of anonym-  
 6350 ous functions,<sup>5</sup> e.g.,

$$\lambda x.\text{LIKES}(x, \text{BRIT}). \quad [12.10]$$

6351 This functional expression is the meaning of the verb phrase *likes Brit*; it takes a single  
 6352 argument, and returns the result of substituting that argument for  $x$  in the expression  
 6353  $\text{LIKES}(x, \text{BRIT})$ . We write this substitution as,

$$(\lambda x.\text{LIKES}(x, \text{BRIT}))@\text{ALEX} = \text{LIKES}(\text{ALEX}, \text{BRIT}), \quad [12.11]$$

6354 with the symbol "@" indicating function application. Function application in the lambda  
 6355 calculus is sometimes called  **$\beta$ -reduction** or  **$\beta$ -conversion**. The expression  $\phi@\psi$  indicates  
 6356 a function application to be performed by  $\beta$ -reduction, and  $\phi(\psi)$  indicates a function or  
 6357 predicate in the final logical form.

6358 Equation 12.11 shows how to obtain the desired semantics for the sentence *Alex likes*  
 6359 *Brit*: by applying the lambda expression  $\lambda x.\text{LIKES}(x, \text{BRIT})$  to the logical constant ALEX.  
 6360 This rule of composition can be specified in a **syntactic-semantic grammar**, in which  
 6361 syntactic productions are paired with semantic operations. For the syntactic production  
 6362  $S \rightarrow NP VP$ , 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,

$$VP.sem = (V.sem)@NP.sem \quad [12.12]$$

$$= (\lambda y.\lambda x.\text{LIKES}(x, y))@(\text{BRIT}) \quad [12.13]$$

$$= \lambda x.\text{LIKES}(x, \text{BRIT}). \quad [12.14]$$

---

<sup>5</sup>Formally, all first-order logic formulas are lambda expressions; in addition, if  $\phi$  is a lambda expression, then  $\lambda x.\phi$  is also a lambda expression. Readers who are familiar with functional programming will recognize lambda expressions from their use in programming languages such as Lisp and Python.

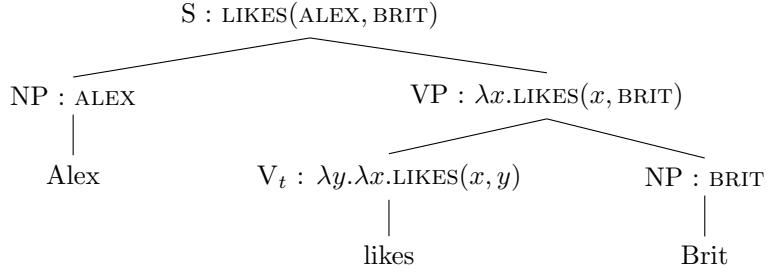


Figure 12.3: Derivation of the semantic representation for *Alex likes Brit* in the grammar  $G_1$ .

S	$\rightarrow$	NP VP	VP.sem@NP.sem
VP	$\rightarrow$	V <sub>t</sub> NP	V <sub>t</sub> .sem@NP.sem
VP	$\rightarrow$	V <sub>i</sub>	V <sub>i</sub> .sem
V <sub>t</sub>	$\rightarrow$	likes	$\lambda y. \lambda x. \text{LIKES}(x, y)$
V <sub>i</sub>	$\rightarrow$	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

6363 Thus, the meaning of the transitive verb *likes* is a lambda expression whose output is  
 6364 *another* lambda expression: it takes  $y$  as an argument to fill in one of the slots in the LIKES  
 6365 relation, and returns a lambda expression that is ready to take an argument to fill in the  
 6366 other slot.<sup>6</sup>

6367 Table 12.1 shows a minimal syntactic-semantic grammar fragment,  $G_1$ . The complete  
 6368 **derivation** of *Alex likes Brit* in  $G_1$  is shown in Figure 12.3. In addition to the transitive  
 6369 verb *likes*, the grammar also includes the intransitive verb *sleeps*; it should be clear how  
 6370 to derive the meaning of sentences like *Alex sleeps*. For verbs that can be either transitive  
 6371 or intransitive, such as *eats*, we would have two terminal productions, one for each sense  
 6372 (terminal productions are also called the **lexical entries**). Indeed, most of the grammar is  
 6373 in the **lexicon** (the terminal productions), since these productions select the basic units of  
 6374 the semantic interpretation.

<sup>6</sup>This can be written in a few different ways. The notation  $\lambda y. x. \text{LIKES}(x, y)$  is a somewhat informal way to indicate a lambda expression that takes two arguments; this would be acceptable in functional programming. Logicians (e.g., Carpenter, 1997) often prefer the more formal notation  $\lambda y. \lambda x. \text{LIKES}(x)(y)$ , indicating that each lambda expression takes exactly one argument.

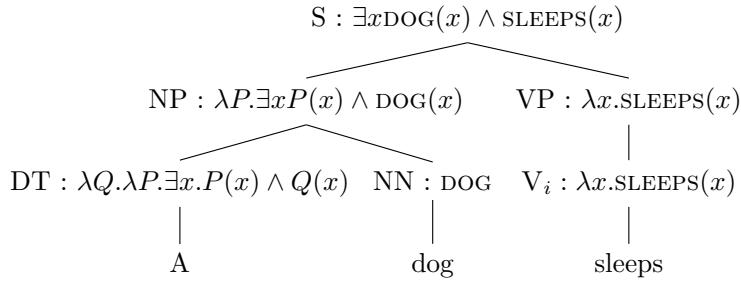


Figure 12.4: Derivation of the semantic representation for *A dog sleeps*, in grammar  $G_2$

### 6375 12.3.2 Quantification

6376 Things get more complicated when we move from sentences about named entities to sen-  
 6377 tences that involve more general noun phrases. Let's consider the example, *A dog sleeps*,  
 6378 which has the meaning  $\exists x \text{DOG}(x) \wedge \text{SLEEPS}(x)$ . Clearly, the DOG relation will be intro-  
 6379 duced by the word *dog*, and the SLEEP relation will be introduced by the word *sleeps*.<sup>7</sup>  
 6380 The existential quantifier  $\exists$  must be introduced by the lexical entry for the determiner *a*.<sup>7</sup>  
 6381 However, this seems problematic for the compositional approach taken in the grammar  
 6382  $G_1$ : if the semantics of the noun phrase *a dog* is an existentially quantified expression, how  
 6383 can it be the argument to the semantics of the verb *sleeps*, which expects an entity? And  
 6384 where does the logical conjunction come from?

6385 There are a few different approaches to handling these issues.<sup>8</sup> We will begin by re-  
 6386 versing the semantic relationship between subject NPs and VPs, so that the production  
 6387  $S \rightarrow \text{NP VP}$  has the semantics  $\text{NP.sem} @ \text{VP.sem}$ : the meaning of the sentence is now the  
 6388 semantics of the noun phrase applied to the verb phrase. The implications of this change  
 6389 are best illustrated by exploring the derivation of the example, shown in Figure 12.4. Let's  
 6390 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

---

<sup>7</sup>Conversely, the sentence *Every dog sleeps* would involve a universal quantifier,  $\forall x \text{DOG}(x) \Rightarrow \text{SLEEPS}(x)$ . The definite article *the* requires more consideration, since *the dog* must refer to some dog which is uniquely identifiable, perhaps from contextual information external to the sentence. Carpenter (1997, pp. 96-100) summarizes recent approaches to handling definite descriptions.

<sup>8</sup>Carpenter (1997) offers an alternative treatment based on combinatory categorial grammar.

adjacent noun, which in this case is DOG. Thus, the noun phrase *a dog* has the semantics,

$$\text{NP.sem} = \text{DET.sem} @ \text{NN.sem} \quad [12.16]$$

$$= (\lambda P. \lambda Q. \exists x P(x) \wedge Q(x)) @ (\text{DOG}) \quad [12.17]$$

$$= \lambda Q. \exists x \text{DOG}(x) \wedge Q(x). \quad [12.18]$$

6391 This is a lambda expression that is expecting another relation,  $Q$ , which will be provided  
 6392 by the verb phrase, SLEEPS. This gives the desired analysis,  $\exists x \text{DOG}(x) \wedge \text{SLEEPS}(x)$ .<sup>9</sup>

6393 If noun phrases like *a dog* are interpreted as lambda expressions, then proper nouns  
 6394 like *Alex* must be treated in the same way. This is achieved by **type-raising** from con-  
 6395 stants to lambda expressions,  $x \Rightarrow \lambda P. P(x)$ . After type-raising, the semantics of *Alex* is  
 6396  $\lambda P. P(\text{ALEX})$  — a lambda expression that expects a relation to tell us something about  
 6397 *ALEX*.<sup>10</sup> Again, make sure you see how the analysis in Figure 12.4 can be applied to the  
 6398 sentence *Alex sleeps*.

6399 Direct objects are handled by applying the same type-raising operation to transitive  
 6400 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,

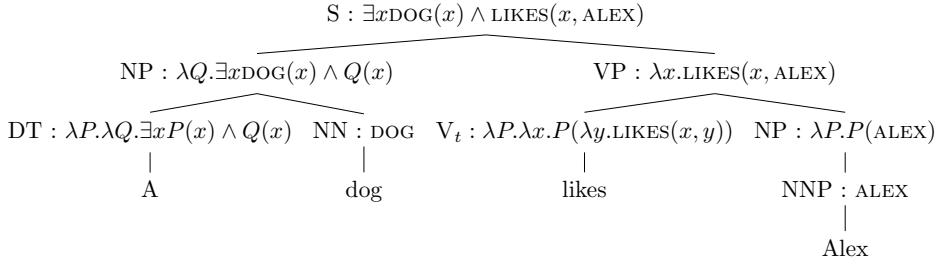
$$\begin{aligned} \text{VP.sem} &= \text{V.sem} @ \text{NP.sem} \\ &= (\lambda P. \lambda x. P(\lambda y. \text{LIKES}(x, y))) @ (\lambda Q. \exists z \text{DOG}(z) \wedge Q(z)) \\ &= \lambda x. (\lambda Q. \exists z \text{DOG}(z) \wedge Q(z)) @ (\lambda y. \text{LIKES}(x, y)) \\ &= \lambda x. \exists z \text{DOG}(z) \wedge (\lambda y. \text{LIKES}(x, y)) @ z \\ &= \lambda x. \exists z \text{DOG}(z) \wedge \text{LIKES}(x, z). \end{aligned}$$

6401 These changes are summarized in the revised grammar  $G_2$ , shown in Table 12.2. Fig-  
 6402 ure 12.5 shows a derivation that involves a transitive verb, an indefinite noun phrase, and  
 6403 a proper noun.

---

<sup>9</sup>When applying  $\beta$ -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  **$\alpha$ -conversion**. For example,  $\lambda x. P(x)$  can be converted to  $\lambda y. P(y)$ , etc.

<sup>10</sup>Compositional semantic analysis is often supported by **type systems**, which make it possible to check whether a given function application is valid. The base types are entities  $e$  and truth values  $t$ . A property, such as DOG, is a function from entities to truth values, so its type is written  $\langle e, t \rangle$ . A transitive verb has type

Figure 12.5: Derivation of the semantic representation for *A dog likes Alex*.

S	$\rightarrow \text{NP VP}$	NP.sem@VP.sem
VP	$\rightarrow \text{V}_t \text{NP}$	$\text{V}_t.\text{sem}@\text{NP.sem}$
VP	$\rightarrow \text{V}_i$	$\text{V}_i.\text{sem}$
NP	$\rightarrow \text{DET NN}$	DET.sem@NN.sem
NP	$\rightarrow \text{NNP}$	$\lambda P. P(\text{NNP.sem})$
DET	$\rightarrow a$	$\lambda P. \lambda Q. \exists x P(x) \wedge Q(x)$
DET	$\rightarrow \text{every}$	$\lambda P. \lambda Q. \forall x (P(x) \Rightarrow Q(x))$
$\text{V}_t$	$\rightarrow \text{likes}$	$\lambda P. \lambda x. P(\lambda y. \text{LIKES}(x, y))$
$\text{V}_i$	$\rightarrow \text{sleeps}$	$\lambda x. \text{SLEEPS}(x)$
NN	$\rightarrow \text{dog}$	DOG
NNP	$\rightarrow \text{Alex}$	ALEX
NNP	$\rightarrow \text{Brit}$	BRIT

Table 12.2:  $G_2$ , a syntactic-semantic context-free grammar fragment, which supports quantified noun phrases

## 6404 12.4 Learning semantic parsers

6405 As with syntactic parsing, any syntactic-semantic grammar with sufficient coverage risks  
 6406 producing many possible analyses for any given sentence. Machine learning is the dom-  
 6407 inant approach to selecting a single analysis. We will focus on algorithms that learn to  
 6408 score logical forms by attaching weights to features of their derivations (Zettlemoyer  
 6409 and Collins, 2005). Alternative approaches include transition-based parsing (Zelle and  
 6410 Mooney, 1996; Misra and Artzi, 2016) and methods inspired by machine translation (Wong  
 6411 and Mooney, 2006). Methods also differ in the form of supervision used for learning,

$\langle e, \langle e, t \rangle \rangle$ : after receiving the first entity (the direct object), it returns a function from entities to truth values, which will be applied to the subject of the sentence. The type-raising operation  $x \Rightarrow \lambda P. P(x)$  corresponds to a change in type from  $e$  to  $\langle \langle e, t \rangle, t \rangle$ : it expects a function from entities to truth values, and returns a truth value.

which can range from complete derivations to much more limited training signals. We will begin with the case of complete supervision, and then consider how learning is still possible even when seemingly key information is missing.

**Datasets** Early work on semantic parsing focused on natural language expressions of geographical database queries, such as *What states border Texas*. The GeoQuery dataset of Zelle and Mooney (1996) was originally coded in prolog, but has subsequently been expanded and converted into the SQL database query language by Popescu et al. (2003) and into first-order logic with lambda calculus by Zettlemoyer and Collins (2005), providing logical forms like  $\lambda x.\text{STATE}(x) \wedge \text{BORDERS}(x, \text{TEXAS})$ . Another early dataset consists of instructions for RoboCup robot soccer teams (Kate et al., 2005). More recent work has focused on broader domains, such as the Freebase database (Bollacker et al., 2008), for which queries have been annotated by Krishnamurthy and Mitchell (2012) and Cai and Yates (2013). Other recent datasets include child-directed speech (Kwiatkowski et al., 2012) and elementary school science exams (Krishnamurthy, 2016).

#### 12.4.1 Learning from derivations

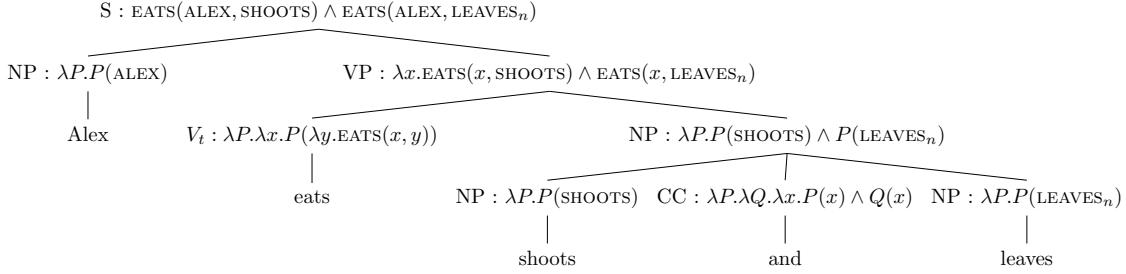
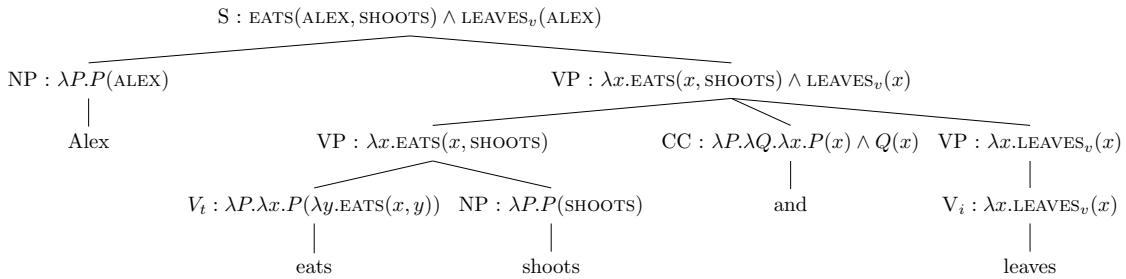
Let  $w^{(i)}$  indicate a sequence of text, and let  $y^{(i)}$  indicate the desired logical form. For example:

$$\begin{aligned} w^{(i)} &= \text{Alex eats shoots and leaves} \\ y^{(i)} &= \text{EATS(ALEX,SHOOTS)} \wedge \text{EATS(ALEX,LEAVES)} \end{aligned}$$

In the standard supervised learning paradigm that was introduced in § 2.2, we first define a feature function,  $f(w, y)$ , and then learn weights on these features, so that  $y^{(i)} = \operatorname{argmax}_y \theta \cdot f(w, y)$ . The weight vector  $\theta$  is learned by comparing the features of the true label  $f(w^{(i)}, y^{(i)})$  against either the features of the predicted label  $f(w^{(i)}, \hat{y})$  (perceptron, support vector machine) or the expected feature vector  $E_{y|w}[f(w^{(i)}, y)]$  (logistic regression).

While this basic framework seems similar to discriminative syntactic parsing, there is a crucial difference. In (context-free) syntactic parsing, the annotation  $y^{(i)}$  contains all of the syntactic productions; indeed, the task of identifying the correct set of productions is identical to the task of identifying the syntactic structure. In semantic parsing, this is not the case: the logical form  $\text{EATS(ALEX,SHOOTS)} \wedge \text{EATS(ALEX,LEAVES)}$  does not reveal the syntactic-semantic productions that were used to obtain it. Indeed, there may be **spurious ambiguity**, so that a single logical form can be reached by multiple derivations. (We previously encountered spurious ambiguity in transition-based dependency parsing, § 11.3.2.)

These ideas can be formalized by introducing an additional variable  $z$ , representing the **derivation** of the logical form  $y$  from the text  $w$ . Assume that the feature function de-

Figure 12.6: Derivation for gold semantic analysis of *Alex eats shoots and leaves*Figure 12.7: Derivation for incorrect semantic analysis of *Alex eats shoots and leaves*

6444 composes across the productions in the derivation,  $f(\mathbf{w}, \mathbf{z}, \mathbf{y}) = \sum_{t=1}^T f(\mathbf{w}, z_t, \mathbf{y})$ , where  
 6445  $z_t$  indicates a single syntactic-semantic production. For example, we might have a feature  
 6446 for the production  $S \rightarrow NP VP : NP.sem@VP.sem$ , as well as for terminal productions  
 6447 like  $NNP \rightarrow Alex : ALEX$ . Under this decomposition, it is possible to compute scores  
 6448 for each semantically-annotated subtree in the analysis of  $\mathbf{w}$ , so that bottom-up parsing  
 6449 algorithms like CKY (§ 10.1) can be applied to find the best-scoring semantic analysis.

6450 Figure 12.6 shows a derivation of the correct semantic analysis of the sentence *Alex*  
 6451 *eats shoots and leaves*, in a simplified grammar in which the plural noun phrases *shoots*  
 6452 and *leaves* are interpreted as logical constants *SHOOTS* and *LEAVES<sub>n</sub>*. Figure 12.7 shows a  
 6453 derivation of an incorrect analysis. Assuming one feature per production, the perceptron  
 6454 update is shown in Table 12.3. From this update, the parser would learn to prefer the  
 6455 noun interpretation of *leaves* over the verb interpretation. It would also learn to prefer  
 6456 noun phrase coordination over verb phrase coordination.

6457 While the update is explained in terms of the perceptron, it would be easy to replace  
 6458 the perceptron with a conditional random field. In this case, the online updates would be  
 6459 based on feature expectations, which can be computed using the inside-outside algorithm  
 6460 (§ 10.6).

$NP_1 \rightarrow NP_2 \ CC \ NP_3$	$(CC.sem @ (NP_2.sem)) @ (NP_3.sem)$	+1
$VP_1 \rightarrow VP_2 \ CC \ VP_3$	$(CC.sem @ (VP_2.sem)) @ (VP_3.sem)$	-1
$NP \rightarrow leaves$	$LEAVES_n$	+1
$VP \rightarrow V_i$	$V_i.sem$	-1
$V_i \rightarrow leaves$	$\lambda x.LEAVES_v$	-1

Table 12.3: Perceptron update for analysis in Figure 12.6 (gold) and Figure 12.7 (predicted)

6461 **12.4.2 Learning from logical forms**

Complete derivations are expensive to annotate, and are rarely available.<sup>11</sup> One solution is to focus on learning from logical forms directly, while treating the derivations as **latent variables** (Zettlemoyer and Collins, 2005). In a conditional probabilistic model over logical forms  $y$  and derivations  $z$ , we have,

$$p(y, z | w) = \frac{\exp(\theta \cdot f(w, z, y))}{\sum_{y', z'} \exp(\theta \cdot f(w, z', y'))}, \quad [12.20]$$

6462 which is the standard log-linear model, applied to the logical form  $y$  and the derivation  
6463  $z$ .

Since the derivation  $z$  unambiguously determines the logical form  $y$ , it may seem silly to model the joint probability over  $y$  and  $z$ . However, since  $z$  is unknown, it can be marginalized out,

$$p(y | w) = \sum_z p(y, z | w). \quad [12.21]$$

The semantic parser can then select the logical form with the maximum log marginal probability,

$$\log \sum_z p(y, z | w) = \log \sum_z \frac{\exp(\theta \cdot f(w, z, y))}{\sum_{y', z'} \exp(\theta \cdot f(w, z', y'))} \quad [12.22]$$

$$\propto \log \sum_z \exp(\theta \cdot f(w, z', y')) \quad [12.23]$$

$$\geq \max_z \theta \cdot f(w, z, y). \quad [12.24]$$

6464 It is impossible to push the log term inside the sum over  $z$ , so our usual linear scoring  
6465 function does not apply. We can recover this scoring function only in approximation, by  
6466 taking the max (rather than the sum) over derivations  $z$ , which provides a lower bound.

---

<sup>11</sup>An exception is the work of Ge and Mooney (2005), who annotate the meaning of each syntactic constituents for several hundred sentences.

Learning can be performed by maximizing the log marginal likelihood,

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^N \log p(\mathbf{y}^{(i)} \mid \mathbf{w}^{(i)}; \boldsymbol{\theta}) \quad [12.25]$$

$$= \sum_{i=1}^N \log \sum_z p(\mathbf{y}^{(i)}, \mathbf{z}^{(i)} \mid \mathbf{w}^{(i)}; \boldsymbol{\theta}). \quad [12.26]$$

6467 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  
6468 initialization.  
6469

The derivative of Equation 12.26 is,

$$\frac{\partial \ell_i}{\partial \boldsymbol{\theta}} = \sum_z p(z \mid \mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) \mathbf{f}(\mathbf{w}, z, \mathbf{y}) - \sum_{z'} p(z', \mathbf{y}' \mid \mathbf{w}; \boldsymbol{\theta}) \mathbf{f}(\mathbf{w}, z', \mathbf{y}') \quad [12.27]$$

$$= E_{z|\mathbf{y}, \mathbf{w}} \mathbf{f}(\mathbf{w}, z, \mathbf{y}) - E_{y, z|\mathbf{w}} \mathbf{f}(\mathbf{w}, z, \mathbf{y}) \quad [12.28]$$

6470 Both expectations can be computed via bottom-up algorithms like inside-outside. Alternatively, we can again maximize rather than marginalize over derivations for an approximating solution. In either case, the first term of the gradient requires us to identify  
6471 derivations  $z$  that are compatible with the logical form  $\mathbf{y}$ . This can be done in a bottom-up dynamic programming algorithm, by having each cell in the table  $t[i, j, X]$  include the  
6472 set of all possible logical forms for  $X \rightsquigarrow \mathbf{w}_{i+1:j}$ . The resulting table may therefore be much  
6473 larger than in syntactic parsing. This can be controlled by using pruning to eliminate intermediate analyses that are incompatible with the final logical form  $\mathbf{y}$  (Zettlemoyer and  
6474 Collins, 2005), or by using beam search and restricting the size of each cell to some fixed  
6475 constant (Liang et al., 2013).  
6476

6477 If we replace each expectation in Equation 12.28 with argmax and then apply stochastic  
6478 gradient descent to learn the weights, we obtain the **latent variable perceptron**, a simple  
6479 and general algorithm for learning with missing data. The algorithm is shown in its most  
6480 basic form in Algorithm 16, but the usual tricks such as averaging and margin loss can  
6481 be applied (Yu and Joachims, 2009). Aside from semantic parsing, the latent variable  
6482 perceptron has been used in tasks such as machine translation (Liang et al., 2006) and  
6483 named entity recognition (Sun et al., 2009). In **latent conditional random fields**, we use  
6484 the full expectations rather than maximizing over the hidden variable. This model has  
6485 also been employed in a range of problems beyond semantic parsing, including parse  
6486 reranking (Koo and Collins, 2005) and gesture recognition (Quattoni et al., 2007).  
6487

### 6490 12.4.3 Learning from denotations

Logical forms are easier to obtain than complete derivations, but the annotation of logical forms still requires considerable expertise. However, it is relatively easy to obtain deno-

**Algorithm 16** Latent variable perceptron

---

```

1: procedure LATENTVARIABLEPERCEPTRON( $\mathbf{w}^{(1:N)}, \mathbf{y}^{(1:N)}$ )
2:    $\theta \leftarrow 0$ 
3:   repeat
4:     Select an instance  $i$ 
5:      $\mathbf{z}^{(i)} \leftarrow \text{argmax}_{\mathbf{z}} \theta \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{z}, \mathbf{y}^{(i)})$ 
6:      $\hat{\mathbf{y}}, \hat{\mathbf{z}} \leftarrow \text{argmax}_{\mathbf{y}', \mathbf{z}'} \theta \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{z}', \mathbf{y}')$ 
7:      $\theta \leftarrow \theta + \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{z}^{(i)}, \mathbf{y}^{(i)}) - \mathbf{f}(\mathbf{w}^{(i)}, \hat{\mathbf{z}}, \hat{\mathbf{y}})$ 
8:   until tired
9:   return  $\theta$ 

```

---

tations for many natural language sentences. For example, in the geography domain, the denotation of a question would be its answer (Clarke et al., 2010; Liang et al., 2013):

**Text** :*What states border Georgia?*  
**Logical form** : $\lambda x.\text{STATE}(x) \wedge \text{BORDER}(x, \text{GEORGIA})$   
**Denotation** :{Alabama, Florida, North Carolina,  
South Carolina, Tennessee}

6491 Similarly, in a robotic control setting, the denotation of a command would be an action or  
6492 sequence of actions (Artzi and Zettlemoyer, 2013). In both cases, the idea is to reward the  
6493 semantic parser for choosing an analysis whose denotation is correct: the right answer to  
6494 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,

$$\ell^{(i)}(\boldsymbol{\theta}) = \log \sum_{\mathbf{y}} v_i(\mathbf{y}) \times p(\mathbf{y} \mid \mathbf{w}; \boldsymbol{\theta}) \quad [12.29]$$

$$= \log \sum_{\mathbf{y}} v_i(\mathbf{y}) \times \sum_{\mathbf{z}} p(\mathbf{y}, \mathbf{z} \mid \mathbf{w}; \boldsymbol{\theta}), \quad [12.30]$$

6495 which sums over all derivations  $\mathbf{z}$  of all valid logical forms,  $\{\mathbf{y} : v_i(\mathbf{y}) = 1\}$ . This cor-  
6496 responds to the log-probability that the semantic parser produces a logical form with a  
6497 valid denotation.

Differentiating with respect to  $\theta$ , we obtain,

$$\frac{\partial \ell^{(i)}}{\partial \theta} = \sum_{\mathbf{y}, \mathbf{z}: v_i(\mathbf{y})=1} p(\mathbf{y}, \mathbf{z} | \mathbf{w}) \mathbf{f}(\mathbf{w}, \mathbf{z}, \mathbf{y}) - \sum_{\mathbf{y}', \mathbf{z}'} p(\mathbf{y}', \mathbf{z}' | \mathbf{w}) \mathbf{f}(\mathbf{w}, \mathbf{z}', \mathbf{y}'), \quad [12.31]$$

which is the usual difference in feature expectations. The positive term computes the expected feature expectations conditioned on the denotation being valid, while the second term computes the expected feature expectations according to the current model, without regard to the ground truth. Large-margin learning formulations are also possible for this problem. For example, Artzi and Zettlemoyer (2013) generate a set of valid and invalid derivations, and then impose a constraint that all valid derivations should score higher than all invalid derivations. This constraint drives a perceptron-like learning rule.

## Additional resources

A key issue not considered here is how to handle **semantic underspecification**: cases in which there are multiple semantic interpretations for a single syntactic structure. Quantifier scope ambiguity is a classic example. Blackburn and Bos (2005) enumerate a number of approaches to this issue, and also provide links between natural language semantics and computational inference techniques. Much of the contemporary research on semantic parsing uses the framework of combinatory categorial grammar (CCG). Carpenter (1997) provides a comprehensive treatment of how CCG can support compositional semantic analysis. Another recent area of research is the semantics of multi-sentence texts. This can be handled with models of **dynamic semantics**, such as dynamic predicate logic (Groenendijk and Stokhof, 1991).

Alternative readings on formal semantics include an “informal” reading from Levy and Manning (2009), and a more involved introduction from Briscoe (2011). To learn more about ongoing research on data-driven semantic parsing, readers may consult the survey article by Liang and Potts (2015), tutorial slides and videos by Artzi and Zettlemoyer (2013),<sup>12</sup> and the source code by Yoav Artzi<sup>13</sup> and Percy Liang.<sup>14</sup>

## Exercises

- Derive the **modus ponens** inference rule, which states that if we know  $\phi \Rightarrow \psi$  and  $\phi$ , then  $\psi$  must be true. The derivation can be performed using the definition of the  $\Rightarrow$  operator and some of the laws provided in § 12.2.1, plus one additional identity:  $\perp \vee \phi = \phi$ .

---

<sup>12</sup>Videos are currently available at <http://yoavartzi.com/tutorial/>

<sup>13</sup><http://yoavartzi.com/spf>

<sup>14</sup><https://github.com/percyliang/sempre>

- 6526     2. Convert the following examples into first-order logic, using the relations CAN-SLEEP,  
 6527     MAKES-NOISE, and BROTHER.
- 6528       • If Abigail makes noise, no one can sleep.  
 6529       • If Abigail makes noise, someone cannot sleep.  
 6530       • None of Abigail's brothers can sleep.  
 6531       • If one of Abigail's brothers makes noise, Abigail cannot sleep.
- 6532     3. Extend the grammar fragment  $G_1$  to include the ditransitive verb *teaches* and the  
 6533     proper noun *Swahili*. Show how to derive the interpretation for the sentence *Alex*  
 6534     *teaches Brit Swahili*, which should be  $\text{TEACHES}(\text{ALEX}, \text{BRIT}, \text{SWAHILI})$ . The grammar  
 6535     need not be in Chomsky Normal Form. For the ditransitive verb, use  $\text{NP}_1$  and  $\text{NP}_2$   
 6536     to indicate the two direct objects.
- 6537     4. Derive the semantic interpretation for the sentence *Alex likes every dog*, using gram-  
 6538     mar fragment  $G_2$ .
- 6539     5. Extend the grammar fragment  $G_2$  to handle adjectives, so that the meaning of *an  
 6540     angry dog* is  $\lambda P. \exists x \text{DOG}(x) \wedge \text{ANGRY}(x) \wedge P(x)$ . Specifically, you should supply the  
 6541     lexical entry for the adjective *angry*, and you should specify the syntactic-semantic  
 6542     productions  $\text{NP} \rightarrow \text{DET } \text{NOM}$ ,  $\text{NOM} \rightarrow \text{JJ } \text{NOM}$ , and  $\text{NOM} \rightarrow \text{NN}$ .
- 6543     6. Extend your answer to the previous question to cover copula constructions with  
 6544     predicative adjectives, such as *Alex is angry*. The interpretation should be  $\text{ANGRY}(\text{ALEX})$ .  
 6545     You should add a verb phrase production  $\text{VP} \rightarrow \text{V}_{\text{cop}} \text{ JJ}$ , and a terminal production  
 6546      $\text{V}_{\text{cop}} \rightarrow \text{is}$ . Show why your grammar extensions result in the correct interpretation.
- 6547     7. In Figure 12.6 and Figure 12.7, we treat the plurals *shoots* and *leaves* as entities. Revise  
 6548      $G_2$  so that the interpretation of *Alex eats leaves* is  $\forall x. (\text{LEAF}(x) \Rightarrow \text{EATS}(\text{ALEX}, x))$ , and  
 6549     show the resulting perceptron update.
- 6547     8. Statements like *every student eats a pizza* have two possible interpretations, depend-  
 6548     ing on quantifier scope:

$$\forall x \exists y \text{PIZZA}(y) \wedge (\text{STUDENT}(x) \Rightarrow \text{EATS}(x, y)) \quad [12.32]$$

$$\exists y \forall x \text{PIZZA}(y) \wedge (\text{STUDENT}(x) \Rightarrow \text{EATS}(x, y)) \quad [12.33]$$

6550     Explain why these interpretations really are different, and modify the grammar  $G_2$   
 6551     so that it can produce both interpretations.

- 6552     9. Derive Equation 12.27.
- 6553     10. In the GeoQuery domain, give a natural language query that has multiple plausible  
 6554     semantic interpretations with the same denotation. List both interpretaions and the  
 6555     denotation.

6556      **Hint:** There are many ways to do this, but one approach involves using toponyms  
6557      (place names) that could plausibly map to several different entities in the model.



6558 

## Chapter 13

6559 

# Predicate-argument semantics

6560 This chapter considers more “lightweight” semantic representations, which discard some  
6561 aspects of first-order logic, but focus on predicate-argument structures. Let’s begin by  
6562 thinking about the semantics of events, with a simple example:

6563 (13.1) Asha gives Boyang a book.

6564 A first-order logical representation of this sentence is,

$$\exists x. \text{BOOK}(x) \wedge \text{GIVE}(\text{ASHA}, \text{BOYANG}, x) \quad [13.1]$$

6565 In this representation, we define variable  $x$  for the book, and we link the strings *Asha* and  
6566 *Boyang* to entities ASHA and BOYANG. Because the action of giving involves a giver, a  
6567 recipient, and a gift, the predicate GIVE must take three arguments.

6568 Now suppose we have additional information about the event:

6569 (13.2) Yesterday, Asha reluctantly gave Boyang a book.

6570 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  $e$ ,

$$\begin{aligned} \exists e, x. & \text{GIVE-EVENT}(e) \wedge \text{GIVER}(e, \text{ASHA}) \wedge \text{GIFT}(e, x) \wedge \text{BOOK}(e, x) \wedge \text{RECIPIENT}(e, \text{BOYANG}) \\ & \wedge \text{TIME}(e, \text{YESTERDAY}) \wedge \text{MANNER}(e, \text{RELUCTANTLY}) \end{aligned}$$

6571 In this way, each argument of the event — the giver, the recipient, the gift — can be rep-  
 6572 resented with a relation of its own, linking the argument to the event  $e$ . The expression  
 6573 GIVER( $e$ , ASHA) says that ASHA plays the **role** of GIVER in the event. This reformulation  
 6574 handles the problem of optional information such as the time or manner of the event,  
 6575 which are called **adjuncts**. Unlike arguments, adjuncts are not a mandatory part of the  
 6576 relation, but under this representation, they can be expressed with additional logical rela-  
 6577 tions that are conjoined to the semantic interpretation of the sentence.<sup>1</sup>

6578 The event semantic representation can be applied to nested clauses, e.g.,

6579 (13.3) Chris sees Asha pay Boyang.

This is done by using the event variable as an argument:

$$\begin{aligned} \exists e_1 \exists e_2 \text{SEE-EVENT}(e_1) \wedge \text{SEER}(e_1, \text{CHRIS}) \wedge \text{SIGHT}(e_1, e_2) \\ \wedge \text{PAY-EVENT}(e_2) \wedge \text{PAYER}(e_2, \text{ASHA}) \wedge \text{PAYEE}(e_2, \text{BOYANG}) \end{aligned} \quad [13.3]$$

6580 As with first-order logic, the goal of event semantics is to provide a representation that  
 6581 generalizes over many surface forms. Consider the following paraphrases of (13.1):

- 6582 (13.4) Asha gives a book to Boyang.
- 6583 (13.5) A book is given to Boyang by Asha.
- 6584 (13.6) A book is given by Asha to Boyang.
- 6585 (13.7) The gift of a book from Asha to Boyang ...

6586 All have the same event semantic meaning as Equation 13.1, but the ways in which the  
 6587 meaning can be expressed are diverse. The final example does not even include a verb:  
 6588 events are often introduced by verbs, but as shown by (13.7), the noun *gift* can introduce  
 6589 the same predicate, with the same accompanying arguments.

6590 **Semantic role labeling** (SRL) is a relaxed form of semantic parsing, in which each  
 6591 semantic role is filled by a set of tokens from the text itself. This is sometimes called  
 6592 “shallow semantics” because, unlike model-theoretic semantic parsing, role fillers need  
 6593 not be symbolic expressions with denotations in some world model. A semantic role  
 6594 labeling system is required to identify all predicates, and then specify the spans of text  
 6595 that fill each role. To give a sense of the task, here is a more complicated example:

- 6596 (13.8) Boyang wants Asha to give him a linguistics book.

---

<sup>1</sup>This 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).

6597 In this example, there are two predicates, expressed by the verbs *want* and *give*. Thus, a  
 6598 semantic role labeler might return the following output:

- 6599 • (PREDICATE : *wants*, WANTED : *Boyang*, DESIRE : *Asha to give him a linguistics book*)  
 6600 • (PREDICATE : *give*, GIVER : *Asha*, RECIPIENT : *him*, GIFT : *a linguistics book*)

6601 *Boyang* and *him* may refer to the same person, but the semantic role labeling is not re-  
 6602 quired to resolve this reference. Other predicate-argument representations, such as **Ab-**  
 6603 **stract Meaning Representation (AMR)**, do require reference resolution. We will return to  
 6604 AMR in § 13.3, but first, let us further consider the definition of semantic roles.

6605 **13.1 Semantic roles**

6606 In event semantics, it is necessary to specify a number of additional logical relations to  
 6607 link arguments to events: GIVER, RECIPIENT, SEER, SIGHT, etc. Indeed, every predicate re-  
 6608 quires a set of logical relations to express its own arguments. In contrast, adjuncts such as  
 6609 TIME and MANNER are shared across many types of events. A natural question is whether  
 6610 it is possible to treat mandatory arguments more like adjuncts, by identifying a set of  
 6611 generic argument types that are shared across many event predicates. This can be further  
 6612 motivated by examples involving related verbs:

- 6613 (13.9) Asha gave Boyang a book.  
 6614 (13.10) Asha loaned Boyang a book.  
 6615 (13.11) Asha taught Boyang a lesson.  
 6616 (13.12) Asha gave Boyang a lesson.

6617 The respective roles of Asha, Boyang, and the book are nearly identical across the first  
 6618 two examples. The third example is slightly different, but the fourth example shows that  
 6619 the roles of GIVER and TEACHER can be viewed as related.

6620 One way to think about the relationship between roles such as GIVER and TEACHER is  
 6621 by enumerating the set of properties that an entity typically possesses when it fulfills these  
 6622 roles: givers and teachers are usually **animate** (they are alive and sentient) and **volitional**  
 6623 (they choose to enter into the action).<sup>2</sup> In contrast, the thing that gets loaned or taught is  
 6624 usually not animate or volitional; furthermore, it is unchanged by the event.

6625 Building on these ideas, **thematic roles** generalize across predicates by leveraging the  
 6626 shared semantic properties of typical role fillers (Fillmore, 1968). For example, in exam-  
 6627 ples (13.9-13.12), Asha plays a similar role in all four sentences, which we will call the

---

<sup>2</sup>There are always exceptions. For example, in the sentence *The C programming language has taught me a lot about perseverance*, the “teacher” is the *The C programming language*, which is presumably not animate or volitional.

	<i>Asha</i>	<i>gave</i>	<i>Boyang</i>	<i>a book</i>
<b>VerbNet</b>	AGENT		RECIPIENT	THEME
<b>PropBank</b>	ARG0: giver		ARG2: entity given to	ARG1: thing given
<b>FrameNet</b>	DONOR		RECIPIENT	THEME
	<i>Asha</i>	<i>taught</i>	<i>Boyang</i>	<i>algebra</i>
<b>VerbNet</b>	AGENT		RECIPIENT	TOPIC
<b>PropBank</b>	ARG0: teacher		ARG2: student	ARG1: subject
<b>FrameNet</b>	TEACHER		STUDENT	SUBJECT

Figure 13.1: Example semantic annotations according to VerbNet, PropBank, and FrameNet

6628 **agent.** This reflects several shared semantic properties: she is the one who is actively and  
 6629 intentionally performing the action, while Boyang is a more passive participant; the book  
 6630 and the lesson would play a different role, as non-animate participants in the event.

6631 Example annotations from three well known systems are shown in Figure 13.1. We  
 6632 will now discuss these systems in more detail.

### 6633 13.1.1 VerbNet

6634 **VerbNet** (Kipper-Schuler, 2005) is a lexicon of verbs, and it includes thirty “core” thematic  
 6635 roles played by arguments to these verbs. Here are some example roles, accompanied by  
 6636 their definitions from the VerbNet Guidelines.<sup>3</sup>

- 6637 • AGENT: “ACTOR in an event who initiates and carries out the event intentionally or  
   6638 consciously, and who exists independently of the event.”
- 6639 • PATIENT: “UNDERGOER in an event that experiences a change of state, location or  
   6640 condition, that is causally involved or directly affected by other participants, and  
   6641 exists independently of the event.”
- 6642 • RECIPIENT: “DESTINATION that is animate”
- 6643 • THEME: “UNDERGOER that is central to an event or state that does not have control  
   6644 over the way the event occurs, is not structurally changed by the event, and/or is  
   6645 characterized as being in a certain position or condition throughout the state.”
- 6646 • TOPIC: “THEME characterized by information content transferred to another partic-  
   6647 ipant.”

<sup>3</sup>[http://verbs.colorado.edu/verb-index/VerbNet\\_Guidelines.pdf](http://verbs.colorado.edu/verb-index/VerbNet_Guidelines.pdf)

6648 VerbNet roles are organized in a hierarchy, so that a TOPIC is a type of THEME, which in  
 6649 turn is a type of UNDERGOER, which is a type of PARTICIPANT, the top-level category.

6650 In addition, VerbNet organizes verb senses into a class hierarchy, in which verb senses  
 6651 that have similar meanings are grouped together. Recall from § 4.2 that multiple meanings  
 6652 of the same word are called **senses**, and that WordNet identifies senses for many English  
 6653 words. VerbNet builds on WordNet, so that verb classes are identified by the WordNet  
 6654 senses of the verbs that they contain. For example, the verb class give-13.1 includes  
 6655 the first WordNet sense of *loan* and the second WordNet sense of *lend*.

6656 Each VerbNet class or subclass takes a set of thematic roles. For example, give-13.1  
 6657 takes arguments with the thematic roles of AGENT, THEME, and RECIPIENT;<sup>4</sup> the pred-  
 6658 icate TEACH takes arguments with the thematic roles AGENT, TOPIC, RECIPIENT, and  
 6659 SOURCE.<sup>5</sup> So according to VerbNet, *Asha* and *Boyang* play the roles of AGENT and RECIP-  
 6660 IENT in the sentences,

6661 (13.13) Asha gave Boyang a book.

6662 (13.14) Asha taught Boyang algebra.

6663 The *book* and *algebra* are both THEMES, but *algebra* is a subcategory of THEME — a TOPIC  
 6664 — because it consists of information content that is given to the receiver.

### 6665 13.1.2 Proto-roles and PropBank

6666 Detailed thematic role inventories of the sort used in VerbNet are not universally accepted.  
 6667 For example, Dowty (1991, pp. 547) notes that “Linguists have often found it hard to agree  
 6668 on, and to motivate, the location of the boundary between role types.” He argues that a  
 6669 solid distinction can be identified between just two **proto-roles**:

6670 **Proto-Agent.** Characterized by volitional involvement in the event or state; sentience  
 6671 and/or perception; causing an event or change of state in another participant; move-  
 6672 ment; exists independently of the event.

6673 **Proto-Patient.** Undergoes change of state; causally affected by another participant; sta-  
 6674 tionary relative to the movement of another participant; does not exist indepen-  
 6675 dently of the event.<sup>6</sup>

---

<sup>4</sup><https://verbs.colorado.edu/verb-index/vn/give-13.1.php>

<sup>5</sup>[https://verbs.colorado.edu/verb-index/vn/transfer\\_mesg-37.1.1.php](https://verbs.colorado.edu/verb-index/vn/transfer_mesg-37.1.1.php)

<sup>6</sup>Reisinger 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.

6676 In the examples in Figure 13.1, Asha has most of the proto-agent properties: in giving  
 6677 the book to Boyang, she is acting volitionally (as opposed to *Boyang got a book from Asha*, in  
 6678 which it is not clear whether Asha gave up the book willingly); she is sentient; she causes  
 6679 a change of state in Boyang; she exists independently of the event. Boyang has some  
 6680 proto-agent properties: he is sentient and exists independently of the event. But he also  
 6681 some proto-patient properties: he is the one who is causally affected and who undergoes  
 6682 change of state. The book that Asha gives Boyang has even fewer of the proto-agent  
 6683 properties: it is not volitional or sentient, and it has no causal role. But it also lacks many  
 6684 of the proto-patient properties: it does not undergo change of state, exists independently  
 6685 of the event, and is not stationary.

6686 The **Proposition Bank**, or PropBank (Palmer et al., 2005), builds on this basic agent-  
 6687 patient distinction, as a middle ground between generic thematic roles and roles that are  
 6688 specific to each predicate. Each verb is linked to a list of numbered arguments, with ARG0  
 6689 as the proto-agent and ARG1 as the proto-patient. Additional numbered arguments are  
 6690 verb-specific. For example, for the predicate TEACH,<sup>7</sup> the arguments are:

- 6691 • ARG0: the teacher
- 6692 • ARG1: the subject
- 6693 • ARG2: the student(s)

6694 Verbs may have any number of arguments: for example, WANT and GET have five, while  
 6695 EAT has only ARG0 and ARG1. In addition to the semantic arguments found in the frame  
 6696 files, roughly a dozen general-purpose **adjuncts** may be used in combination with any  
 6697 verb. These are shown in Table 13.1.

6698 PropBank-style semantic role labeling is annotated over the entire Penn Treebank. This  
 6699 annotation includes the sense of each verbal predicate, as well as the argument spans.

### 6700 13.1.3 FrameNet

6701 Semantic **frames** are descriptions of situations or events. Frames may be *evoked* by one  
 6702 of their **lexical units** (often a verb, but not always), and they include some number of  
 6703 **frame elements**, which are like roles (Fillmore, 1976). For example, the act of teaching  
 6704 is a frame, and can be evoked by the verb *taught*; the associated frame elements include  
 6705 the teacher, the student(s), and the subject being taught. Frame semantics has played a  
 6706 significant role in the history of artificial intelligence, in the work of Minsky (1974) and  
 6707 Schank and Abelson (1977). In natural language processing, the theory of frame semantics  
 6708 has been implemented in **FrameNet** (Fillmore and Baker, 2009), which consists of a lexicon

---

<sup>7</sup><http://verbs.colorado.edu/propbank/framesets-english-aliases/teach.html>

---

TMP	time	<i>Boyang ate a bagel</i> [AM-TMP <i>yesterday</i> ].
LOC	location	<i>Asha studies in</i> [AM-LOC <i>Stuttgart</i> ]
MOD	modal verb	<i>Asha</i> [AM-MOD <i>will</i> ] <i>study in Stuttgart</i>
ADV	general purpose	[AM-ADV <i>Luckily</i> ], <i>Asha knew algebra</i> .
MNR	manner	<i>Asha ate</i> [AM-MNR <i>aggressively</i> ].
DIS	discourse connective	[AM-DIS <i>However</i> ], <i>Asha prefers algebra</i> .
PRP	purpose	<i>Barry studied</i> [AM-PRP <i>to pass the bar</i> ].
DIR	direction	<i>Workers dumped burlap sacks</i> [AM-DIR <i>into a bin</i> ].
NEG	negation	<i>Asha does</i> [AM-NEG <i>not</i> ] <i>speak Albanian</i> .
EXT	extent	<i>Prices increased</i> [AM-EXT <i>4%</i> ].
CAU	cause	<i>Boyang returned the book</i> [AM-CAU <i>because it was overdue</i> ].

---

Table 13.1: PropBank adjuncts (Palmer et al., 2005), sorted by frequency in the corpus

6709 of roughly 1000 frames, and a corpus of more than 200,000 “exemplar sentences,” in which  
 6710 the frames and their elements are annotated.<sup>8</sup>

6711 Rather than seeking to link semantic roles such as TEACHER and GIVER into the-  
 6712 matic roles such as AGENT, FrameNet aggressively groups verbs into frames, and links  
 6713 semantically-related roles across frames. For example, the following two sentences would  
 6714 be annotated identically in FrameNet:

6715 (13.15) Asha taught Boyang algebra.

6716 (13.16) Boyang learned algebra from Asha.

6717 This is because *teach* and *learn* are both lexical units in the EDUCATION-TEACHING frame.  
 6718 Furthermore, roles can be shared even when the frames are distinct, as in the following  
 6719 two examples:

6720 (13.17) Asha gave Boyang a book.

6721 (13.18) Boyang got a book from Asha.

6722 The GIVING and GETTING frames both have RECIPIENT and THEME elements, so Boyang  
 6723 and the book would play the same role. Asha’s role is different: she is the DONOR in the  
 6724 GIVING frame, and the SOURCE in the GETTING frame. FrameNet makes extensive use of  
 6725 multiple inheritance to share information across frames and frame elements: for example,  
 6726 the COMMERCE-SELL and LENDING frames inherit from GIVING frame.

<sup>8</sup>Current details and data can be found at <https://framenet.icsi.berkeley.edu/>

## 6727 13.2 Semantic role labeling

6728 The task of semantic role labeling is to identify the parts of the sentence comprising the  
 6729 semantic roles. In English, this task is typically performed on the PropBank corpus, with  
 6730 the goal of producing outputs in the following form:

6731 (13.19) [ARG0 Asha] [GIVE.01 gave] [ARG2 Boyang's mom] [ARG1 a book] [AM-TMP yesterday].

6732 Note that a single sentence may have multiple verbs, and therefore a given word may be  
 6733 part of multiple role-fillers:

6734 (13.20) [ARG0 Asha] [WANT.01 wanted]  
 Asha            wanted

6735            [ARG1 Boyang to give her the book].  
 [ARG0 Boyang] [GIVE.01 to give] [ARG2 her] [ARG1 the book].

### 6736 13.2.1 Semantic role labeling as classification

6737 PropBank is annotated on the Penn Treebank, and annotators used phrasal constituents  
 6738 ( $\S$  9.2.2) to fill the roles. PropBank semantic role labeling can be viewed as the task of as-  
 6739 signing to each phrase a label from the set  $\mathcal{R} = \{\emptyset, \text{PRED}, \text{ARG0}, \text{ARG1}, \text{ARG2}, \dots, \text{AM-LOC}, \text{AM-TMP}, \dots\}$ ,  
 6740 with respect to each predicate. If we treat semantic role labeling as a classification prob-  
 6741 lem, we obtain the following functional form:

$$\hat{y}_{(i,j)} = \underset{y}{\operatorname{argmax}} \psi(\mathbf{w}, y, i, j, \rho, \tau), \quad [13.4]$$

6742 where,

- 6743 •  $(i, j)$  indicates the span of a phrasal constituent  $(w_{i+1}, w_{i+2}, \dots, w_j)$ ;<sup>9</sup>
- 6744 •  $\mathbf{w}$  represents the sentence as a sequence of tokens;
- 6745 •  $\rho$  is the index of the predicate verb in  $\mathbf{w}$ ;
- 6746 •  $\tau$  is the structure of the phrasal constituent parse of  $\mathbf{w}$ .

6747 Early work on semantic role labeling focused on discriminative feature-based models,  
 6748 where  $\psi(\mathbf{w}, y, i, j, \rho, \tau) = \theta \cdot f(\mathbf{w}, y, i, j, \rho, \tau)$ . Table 13.2 shows the features used in a sem-  
 6749 inal paper on FrameNet semantic role labeling (Gildea and Jurafsky, 2002). By 2005 there

---

<sup>9</sup>PropBank 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.

---

<b>Predicate lemma and POS tag</b>	The lemma of the predicate verb and its part-of-speech tag
<b>Voice</b>	Whether the predicate is in active or passive voice, as determined by a set of syntactic patterns for identifying passive voice constructions
<b>Phrase type</b>	The constituent phrase type for the proposed argument in the parse tree, e.g. NP, PP
<b>Headword and POS tag</b>	The head word of the proposed argument and its POS tag, identified using the Collins (1997) rules
<b>Position</b>	Whether the proposed argument comes before or after the predicate in the sentence
<b>Syntactic path</b>	The set of steps on the parse tree from the proposed argument to the predicate (described in detail in the text)
<b>Subcategorization</b>	The syntactic production from the first branching node above the predicate. For example, in Figure 13.2, the subcategorization feature around <i>taught</i> would be VP → VBD NP PP.

---

Table 13.2: Features used in semantic role labeling by Gildea and Jurafsky (2002).

6750 were several systems for PropBank semantic role labeling, and their approaches and fea-  
 6751 ture sets are summarized by Carreras and Márquez (2005). Typical features include: the  
 6752 phrase type, head word, part-of-speech, boundaries, and neighbors of the proposed argu-  
 6753 ment  $w_{i+1:j}$ ; the word, lemma, part-of-speech, and voice of the verb  $w_\rho$  (active or passive),  
 6754 as well as features relating to its frameset; the distance and path between the verb and  
 6755 the proposed argument. In this way, semantic role labeling systems are high-level “con-  
 6756 sumers” in the NLP stack, using features produced from lower-level components such as  
 6757 part-of-speech taggers and parsers. More comprehensive feature sets are enumerated by  
 6758 Das et al. (2014) and Täckström et al. (2015).

6759 A particularly powerful class of features relate to the **syntactic path** between the ar-  
 6760 gument and the predicate. These features capture the sequence of moves required to get  
 6761 from the argument to the verb by traversing the phrasal constituent parse of the sentence.  
 6762 The idea of these features is to capture syntactic regularities in how various arguments  
 6763 are realized. Syntactic path features are best illustrated by example, using the parse tree  
 6764 in Figure 13.2:

- 6765 • The path from *Asha* to the verb *taught* is NNP↑NP↑S↓VP↓VBD. The first part of  
 6766 the path, NNP↑NP↑S, means that we must travel up the parse tree from the NNP  
 6767 tag (proper noun) to the S (sentence) constituent. The second part of the path,  
 6768 S↓VP↓VBD, means that we reach the verb by producing a VP (verb phrase) from

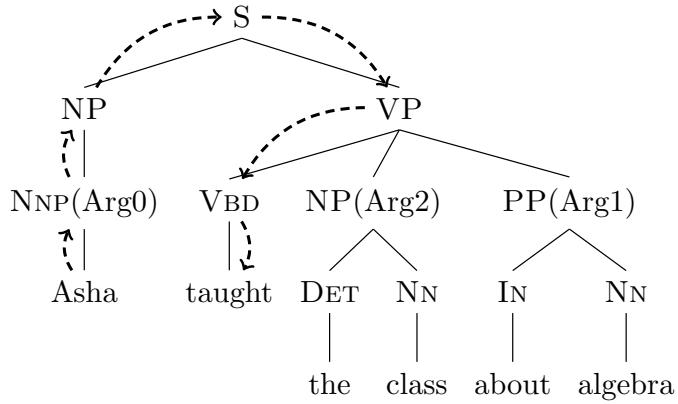


Figure 13.2: Semantic role labeling on the phrase-structure parse tree for a sentence. The dashed line indicates the syntactic path from *Asha* to the predicate verb *taught*.

6769 the S constituent, and then by producing a VBD (past tense verb). This feature is  
 6770 consistent with *Asha* being in subject position, since the path includes the sentence  
 6771 root S.

- 6772 • The path from *the class* to *taught* is NP↑VP↓VBD. This is consistent with *the class*  
 6773 being in object position, since the path passes through the VP node that dominates  
 6774 the verb *taught*.

6775 Because there are many possible path features, it can also be helpful to look at smaller  
 6776 parts: for example, the upward and downward parts can be treated as separate features;  
 6777 another feature might consider whether S appears anywhere in the path.

6778 Rather than using the constituent parse, it is also possible to build features from the  
 6779 **dependency path** between the head word of each argument and the verb (Pradhan et al.,  
 6780 2005). Using the Universal Dependency part-of-speech tagset and dependency relations (Nivre  
 6781 et al., 2016), the dependency path from *Asha* to *taught* is PROPN  $\xleftarrow[\text{NSUBJ}]{} \text{VERB}$ , because *taught*  
 6782 is the head of a relation of type  $\xleftarrow[\text{NSUBJ}]{} \text{VERB}$  with *Asha*. Similarly, the dependency path from *class*  
 6783 to *taught* is NOUN  $\xleftarrow[\text{DOBJ}]{} \text{VERB}$ , because *class* heads the noun phrase that is a direct object of  
 6784 *taught*. A more interesting example is *Asha wanted to teach the class*, where the path from  
 6785 *Asha* to *teach* is PROPN  $\xleftarrow[\text{NSUBJ}]{} \text{VERB} \rightarrow[\text{XCOMP}] \text{VERB}$ . The right-facing arrow in second relation  
 6786 indicates that *wanted* is the head of its XCOMP relation with *teach*.

6787 **13.2.2 Semantic role labeling as constrained optimization**

6788 A potential problem with treating SRL as a classification problem is that there are a num-  
 6789 ber of sentence-level **constraints**, which a classifier might violate.

- 6790 • For a given verb, there can be only one argument of each type (ARG0, ARG1, etc.)  
 6791 • Arguments cannot overlap. This problem arises when we are labeling the phrases  
 6792 in a constituent parse tree, as shown in Figure 13.2: if we label the PP *about algebra*  
 6793 as an argument or adjunct, then its children *about* and *algebra* must be labeled as  $\emptyset$ .  
 6794 The same constraint also applies to the syntactic ancestors of this phrase.

6795 These constraints introduce dependencies across labeling decisions. In structure pre-  
 6796 diction problems such as sequence labeling and parsing, such dependencies are usually  
 6797 handled by defining a scoring over the entire structure,  $\mathbf{y}$ . Efficient inference requires  
 6798 that the global score decomposes into local parts: for example, in sequence labeling, the  
 6799 scoring function decomposes into scores of pairs of adjacent tags, permitting the applica-  
 6800 tion of the Viterbi algorithm for inference. But the constraints that arise in semantic role  
 6801 labeling are less amenable to local decomposition.<sup>10</sup> We therefore consider **constrained**  
 6802 **optimization** as an alternative solution.

Let the set  $\mathcal{C}(\tau)$  refer to all labelings that obey the constraints introduced by the parse  $\tau$ . The semantic role labeling problem can be reformulated as a constrained optimization over  $\mathbf{y} \in \mathcal{C}(\tau)$ ,

$$\begin{aligned} \max_{\mathbf{y}} \quad & \sum_{(i,j) \in \tau} \psi(\mathbf{w}, y_{i,j}, i, j, \rho, \tau) \\ \text{s.t. } \quad & \mathbf{y} \in \mathcal{C}(\tau). \end{aligned} \quad [13.5]$$

6803 In this formulation, the objective (shown on the first line) is a separable function of each  
 6804 individual labeling decision, but the constraints (shown on the second line) apply to the  
 6805 overall labeling. The sum  $\sum_{(i,j) \in \tau}$  indicates that we are summing over all constituent  
 6806 spans in the parse  $\tau$ . The expression s.t. in the second line means that we maximize the  
 6807 objective *subject to* the constraint  $\mathbf{y} \in \mathcal{C}(\tau)$ .

6808 A number of practical algorithms exist for restricted forms of constrained optimiza-  
 6809 tion. One such restricted form is **integer linear programming**, in which the objective and  
 6810 constraints are linear functions of integer variables. To formulate SRL as an integer linear  
 6811 program, we begin by rewriting the labels as a set of binary variables  $\mathbf{z} = \{z_{i,j,r}\}$  (Pun-  
 6812 yakanok et al., 2008),

$$z_{i,j,r} = \begin{cases} 1, & y_{i,j} = r \\ 0, & \text{otherwise,} \end{cases} \quad [13.6]$$

---

<sup>10</sup>Dynamic 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.

6813 where  $r \in \mathcal{R}$  is a label in the set  $\{\text{ARG0}, \text{ARG1}, \dots, \text{AM-LOC}, \dots, \emptyset\}$ . Thus, the variables  
 6814  $\mathbf{z}$  are a binarized version of the semantic role labeling  $\mathbf{y}$ .

The objective can then be formulated as a linear function of  $\mathbf{z}$ .

$$\sum_{(i,j) \in \tau} \psi(\mathbf{w}, y_{i,j}, i, j, \rho, \tau) = \sum_{i,j,r} \psi(\mathbf{w}, r, i, j, \rho, \tau) \times z_{i,j,r}, \quad [13.7]$$

6815 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,

$$\forall r \neq \emptyset, \quad \sum_{(i,j) \in \tau} z_{i,j,r} \leq 1. \quad [13.8]$$

6816 Recall that  $z_{i,j,r} = 1$  iff the span  $(i, j)$  has label  $r$ ; this constraint says that for each possible  
 6817 label  $r \neq \emptyset$ , there can be at most one  $(i, j)$  such that  $z_{i,j,r} = 1$ . Rewriting this constraint  
 6818 can be written in the form  $\mathbf{A}\mathbf{z} \leq \mathbf{b}$ , as you will find if you complete the exercises at the  
 6819 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:

$$\forall (i, j) \in \tau, \quad \sum_{(i', j') \in \tau} \sum_{r \neq \emptyset} o((i, j), (i', j')) \times z_{i',j',r} \leq 1, \quad [13.9]$$

6820 where  $o((i, j), (i, j)) = 1$ .

In summary, the semantic role labeling problem can thus be rewritten as the following integer linear program,

$$\max_{\mathbf{z} \in \{0,1\}^{|\tau|}} \quad \sum_{(i,j) \in \tau} \sum_{r \in \mathcal{R}} z_{i,j,r} \psi_{i,j,r} \quad [13.10]$$

$$s.t. \quad \forall r \neq \emptyset, \quad \sum_{(i,j) \in \tau} z_{i,j,r} \leq 1. \quad [13.11]$$

$$\forall (i, j) \in \tau, \quad \sum_{(i', j') \in \tau} \sum_{r \neq \emptyset} o((i, j), (i', j')) \times z_{i',j',r} \leq 1. \quad [13.12]$$

6821 **Learning with constraints** Learning can be performed in the context of constrained op-  
 6822 timization using the usual perceptron or large-margin classification updates. Because  
 6823 constrained inference is generally more time-consuming, a key question is whether it is  
 6824 necessary to apply the constraints during learning. Chang et al. (2008) find that better per-  
 6825 formance can be obtained by learning *without* constraints, and then applying constraints  
 6826 only when using the trained model to predict semantic roles for unseen data.

6827 **How important are the constraints?** Das et al. (2014) find that an unconstrained, classification-  
 6828 based method performs nearly as well as constrained optimization for FrameNet parsing;  
 6829 while it commits many violations of the “no-overlap” constraint, the overall  $F_1$  score is  
 6830 less than one point worse than the score at the constrained optimum. Similar results  
 6831 were obtained for PropBank semantic role labeling by Punyakanok et al. (2008). He et al.  
 6832 (2017) find that constrained inference makes a bigger impact if the constraints are based  
 6833 on manually-labeled “gold” syntactic parses. This implies that errors from the syntac-  
 6834 tic parser may limit the effectiveness of the constraints. Punyakanok et al. (2008) hedge  
 6835 against parser error by including constituents from several different parsers; any con-  
 6836 stituent can be selected from any parse, and additional constraints ensure that overlap-  
 6837 ping constituents are not selected.

6838 **Implementation** Integer linear programming solvers such as `glpk`,<sup>11</sup> `cplex`,<sup>12</sup> and `Gurobi`<sup>13</sup>  
 6839 allow inequality constraints to be expressed directly in the problem definition, rather than  
 6840 in the matrix form  $\mathbf{A}z \leq b$ . The time complexity of integer linear programming is theoreti-  
 6841 cally exponential in the number of variables  $|z|$ , but in practice these off-the-shelf solvers  
 6842 obtain good solutions efficiently. Das et al. (2014) report that the `cplex` solver requires 43  
 6843 seconds to perform inference on the FrameNet test set, which contains 4,458 predicates.

6844 Recent work has shown that many constrained optimization problems in natural lan-  
 6845 guage processing can be solved in a highly parallelized fashion, using optimization tech-  
 6846 niques such as **dual decomposition**, which are capable of exploiting the underlying prob-  
 6847 lem structure (Rush et al., 2010). Das et al. (2014) apply this technique to FrameNet se-  
 6848 mantic role labeling, obtaining an order-of-magnitude speedup over `cplex`.

### 6849 13.2.3 Neural semantic role labeling

6850 Neural network approaches to SRL have tended to treat it as a sequence labeling task,  
 6851 using a labeling scheme such as the **BIO notation**, which we previously saw in named  
 6852 entity recognition (§ 8.3). In this notation, the first token in a span of type ARG1 is labeled

---

<sup>11</sup><https://www.gnu.org/software/glpk/>

<sup>12</sup><https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>

<sup>13</sup><http://www.gurobi.com/>

6853 B-ARG1; all remaining tokens in the span are *inside*, and are therefore labeled I-ARG1.  
 6854 Tokens outside any argument are labeled O. For example:

- 6855 (13.21) *Asha taught Boyang 's mom about algebra*  
 B-ARG0 PRED B-ARG2 I-ARG2 I-ARG2 B-ARG1 I-ARG1

Recurrent neural networks are a natural approach to this tagging task. For example, Zhou and Xu (2015) apply a deep bidirectional multilayer LSTM (see § 7.6) to PropBank semantic role labeling. In this model, each bidirectional LSTM serves as input for another, higher-level bidirectional LSTM, allowing complex non-linear transformations of the original input embeddings,  $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_M]$ . The hidden state of the final LSTM is  $\mathbf{Z}^{(K)} = [\mathbf{z}_1^{(K)}, \mathbf{z}_2^{(K)}, \dots, \mathbf{z}_M^{(K)}]$ . The “emission” score for each tag  $Y_m = y$  is equal to the inner product  $\theta_y \cdot \mathbf{z}_m^{(K)}$ , and there is also a transition score for each pair of adjacent tags. The complete model can be written,

$$\mathbf{Z}^{(1)} = \text{BiLSTM}(\mathbf{X}) \quad [13.13]$$

$$\mathbf{Z}^{(i)} = \text{BiLSTM}(\mathbf{Z}^{(i-1)}) \quad [13.14]$$

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}} \sum_{m=1}^M \Theta^{(y)} \mathbf{z}_m^{(K)} + \psi_{y_{m-1}, y_m}. \quad [13.15]$$

6856 Note that the final step maximizes over the entire labeling  $\mathbf{y}$ , and includes a score for  
 6857 each tag transition  $\psi_{y_{m-1}, y_m}$ . This combination of LSTM and pairwise potentials on tags  
 6858 is an example of an **LSTM-CRF**. The maximization over  $\mathbf{y}$  is performed by the Viterbi  
 6859 algorithm.

6860 This model strongly outperformed alternative approaches at the time, including con-  
 6861 strained decoding and convolutional neural networks.<sup>14</sup> More recent work has combined  
 6862 recurrent neural network models with constrained decoding, using the  $A^*$  search algo-  
 6863 rithm to search over labelings that are feasible with respect to the constraints (He et al.,  
 6864 2017). This yields small improvements over the method of Zhou and Xu (2015). He et al.  
 6865 (2017) obtain larger improvements by creating an **ensemble** of SRL systems, each trained  
 6866 on an 80% subsample of the corpus. The average prediction across this ensemble is more  
 6867 robust than any individual model.

### 6868 13.3 Abstract Meaning Representation

6869 Semantic role labeling transforms the task of semantic parsing to a labeling task. Consider  
 6870 the sentence,

---

<sup>14</sup>The 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))
```

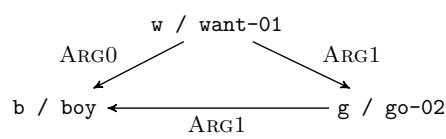


Figure 13.3: Two views of the AMR representation for the sentence *The boy wants to go.*

6871 (13.22) The boy wants to go.

6872 The PropBank semantic role labeling analysis is:

6873 • (PREDICATE : *wants*, ARG0 : *the boy*, ARG1 : *to go*)

6874 • (PREDICATE : *go*, ARG1 : *the boy*)

6875 The **Abstract Meaning Representation (AMR)** unifies this analysis into a graph structure, in which each node is a **variable**, and each edge indicates a **concept** (Banarescu et al., 2013). This can be written in two ways, as shown in Figure 13.3. On the left is the 6876 PENMAN notation (Matthiessen and Bateman, 1991), in which each set of parentheses 6877 introduces a variable. Each variable is an **instance** of a concept, which is indicated with 6878 the slash notation: for example, *w / want-01* indicates that the variable *w* is an instance 6879 of the concept *want-01*, which in turn refers to the PropBank frame for the first sense 6880 of the verb *want*. Relations are introduced with colons: for example, *:ARG0 (b / boy)* 6881 indicates a relation of type *ARG0* with the newly-introduced variable *b*. Variables can be 6882 reused, so that when the variable *b* appears again as an argument to *g*, it is understood to 6883 refer to the same boy in both cases. This arrangement is indicated compactly in the graph 6884 structure on the right, with edges indicating concepts. 6885  
6886

6887 One way in which AMR differs from PropBank-style semantic role labeling is that it 6888 reifies each entity as a variable: for example, *the boy* in (13.22) is reified in the variable 6889 *b*, which is reused as *ARG0* in its relationship with *w / want-01*, and as *ARG1* in its 6890 relationship with *g / go-02*. Reifying entities as variables also makes it possible to 6891 represent the substructure of noun phrases more explicitly. For example, *Asha borrowed* 6892 *the algebra book* would be represented as:

```
6893 (b / borrow-01
  6894   :ARG0 (p / person
  6895     :name (n / name
  6896       :op1 "Asha"))
  6897   :ARG1 (b2 / book
  6898     :topic (a / algebra)))
```

6899 This indicates that the variable *p* is a person, whose name is the variable *n*; that name  
 6900 has one token, the string *Asha*. Similarly, the variable *b2* is a book, and the topic of *b2*  
 6901 is a variable *a* whose type is algebra. The relations name and topic are examples of  
 6902 **non-core roles**, which are similar to adjunct modifiers in PropBank. However, AMR’s  
 6903 inventory is more extensive, including more than 70 non-core roles, such as negation,  
 6904 time, manner, frequency, and location. Lists and sequences — such as the list of tokens in  
 6905 a name — are described using the roles *op1*, *op2*, etc.

6906 Another feature of AMR is that a semantic predicate can be introduced by any syntac-  
 6907 tic element, as in the following examples from Banarescu et al. (2013):

- 6908 (13.23) The boy destroyed the room.
- 6909 (13.24) the destruction of the room by the boy ...
- 6910 (13.25) the boy’s destruction of the room ...

6911 All these examples have the same semantics in AMR,

```
6912 (d / destroy-01
6913   :ARG0 (b / boy)
6914   :ARG1 (r / room))
```

6915 The noun *destruction* is linked to the verb *destroy*, which is captured by the PropBank  
 6916 frame *destroy-01*. This can happen with adjectives as well: in the phrase *the attractive*  
 6917 *spy*, the adjective *attractive* is linked to the PropBank frame *attract-01*:

```
6918 (s / spy
6919   :ARG0-of (a / attract-01))
```

6920 In this example, *ARG0-of* is an **inverse relation**, indicating that *s* is the *ARG0* of the  
 6921 predicate *a*. Inverse relations make it possible for all AMR parses to have a single root  
 6922 concept, which should be the **focus** of the utterance.

6923 While AMR goes farther than semantic role labeling, it does not link semantically-  
 6924 related frames such as buy/sell (as FrameNet does), does not handle quantification (as  
 6925 first-order predicate calculus does), and makes no attempt to handle noun number and  
 6926 verb tense (as PropBank does). A recent survey by Abend and Rappoport (2017) situ-  
 6927 ates AMR with respect to several other semantic representation schemes. Other linguistic  
 6928 features of AMR are summarized in the original paper (Banarescu et al., 2013) and the  
 6929 tutorial slides by Schneider et al. (2015).

### 6930 13.3.1 AMR Parsing

6931 Abstract Meaning Representation is not a labeling of the original text — unlike PropBank  
6932 semantic role labeling, and most of the other tagging and parsing tasks that we have  
6933 encountered thus far. The AMR for a given sentence may include multiple concepts for  
6934 single words in the sentence: as we have seen, the sentence *Asha likes algebra* contains both  
6935 person and name concepts for the word *Asha*. Conversely, words in the sentence may not  
6936 appear in the AMR: in *Boyang made a tour of campus*, the **light verb** *make* would not appear  
6937 in the AMR, which would instead be rooted on the predicate *tour*. As a result, AMR  
6938 is difficult to parse, and even evaluating AMR parsing involves considerable algorithmic  
6939 complexity (Cai and Yates, 2013).

6940 A further complexity is that AMR labeled datasets do not explicitly show the **alignment**  
6941 between the AMR annotation and the words in the sentence. For example, the link  
6942 between the word *wants* and the concept *want-01* is not annotated. To acquire training  
6943 data for learning-based parsers, it is therefore necessary to first perform an alignment  
6944 between the training sentences and their AMR parses. Flanigan et al. (2014) introduce a  
6945 rule-based parser, which links text to concepts through a series of increasingly high-recall  
6946 steps.

6947 **Graph-based parsing** One family of approaches to AMR parsing is similar to the graph-  
6948 based methods that we encountered in syntactic dependency parsing (chapter 11). For  
6949 these systems (Flanigan et al., 2014), parsing is a two-step process:

- 6950 1. **Concept identification** (Figure 13.4a). This involves constructing concept subgraphs  
6951 for individual words or spans of adjacent words. For example, in the sentence,  
6952 *Asha likes algebra*, we would hope to identify the minimal subtree including just the  
6953 concept *like-01* for the word *like*, and the subtree (*p / person :name (n /*  
6954 *name :op1 Asha)*) for the word *Asha*.
- 6955 2. **Relation identification** (Figure 13.4b). This involves building a directed graph over  
6956 the concepts, where the edges are labeled by the relation type. AMR imposes a  
6957 number of constraints on the graph: all concepts must be included, the graph must  
6958 be **connected** (there must be a path between every pair of nodes in the undirected  
6959 version of the graph), and every node must have at most one outgoing edge of each  
6960 type.

6961 Both of these problems are solved by structure prediction. Concept identification re-  
6962 quires simultaneously segmenting the text into spans, and labeling each span with a graph  
6963 fragment containing one or more concepts. This is done by computing a set of features  
6964 for each candidate span *s* and concept labeling *c*, and then returning the labeling with the  
6965 highest overall score.

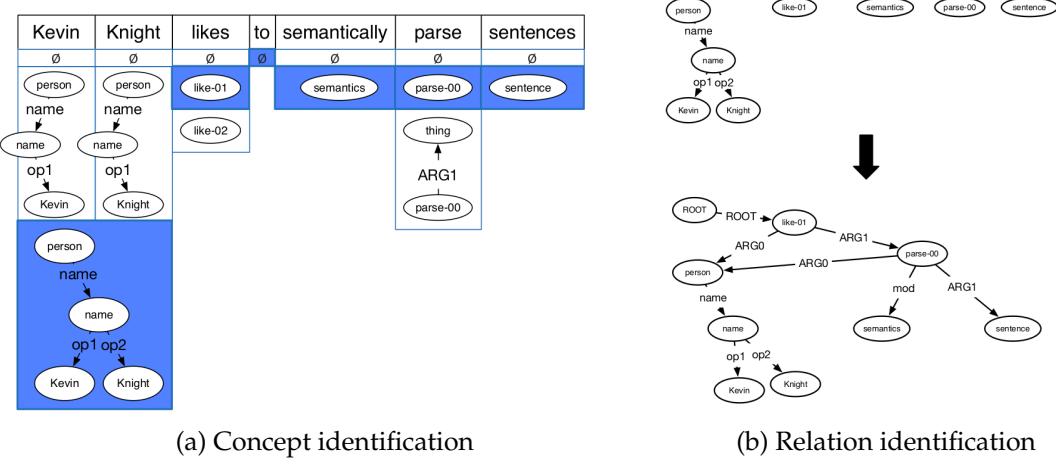


Figure 13.4: Subtasks for Abstract Meaning Representation parsing, from Schneider et al. (2015). [todo: permission]

6966      Relation identification can be formulated as search for the maximum spanning sub-  
 6967      graph, under a set of constraints. Each labeled edge has a score, which is computed  
 6968      from features of the concepts. We then search for the set of labeled edges that maximizes  
 6969      the sum of these scores, under the constraint that the resulting graph is a well-formed  
 6970      AMR (Flanigan et al., 2014). This constrained search can be performed by optimization  
 6971      techniques such as integer linear programming, as described in § 13.2.2.

6972      **Transition-based parsing** In many cases, AMR parses are structurally similar to syn-  
 6973      tactic dependency parses. Figure 13.5 shows one such example. This motivates an alter-  
 6974      native approach to AMR parsing: modify the syntactic dependency parse until it looks  
 6975      like a good AMR parse. Wang et al. (2015) propose a transition-based method, based on  
 6976      incremental modifications to the syntactic dependency tree (transition-based dependency  
 6977      parsing is discussed in § 11.3). At each step, the parser performs an action: for example,  
 6978      adding an AMR relation label to the current dependency edge, swapping the direction of  
 6979      a syntactic dependency edge, or cutting an edge and reattaching the orphaned subtree to  
 6980      a new parent. The overall system is trained as a classifier, learning to choose the action as  
 6981      would be given by an **oracle** that is capable of reproducing the ground-truth parse.

## 6982      13.4 Applications of Predicate-Argument Semantics

6983      **Question answering** Factoid questions have answers that are single words or phrases,  
 6984      such as *who discovered prions?*, *where was Barack Obama born?*, and *in what year did the Knicks*  
 6985      *last win the championship?* Semantic role labeling can be used to answer such questions,

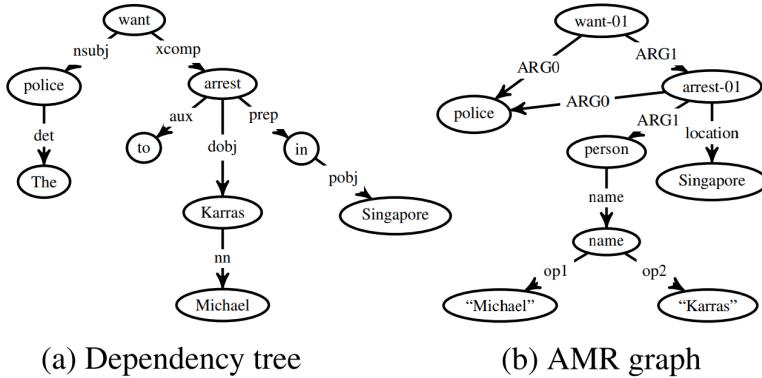


Figure 13.5: Syntactic dependency parse and AMR graph for the sentence *The police want to arrest Michael Karras in Singapore* (borrowed from Wang et al. (2015)) [todo: permission]

6986 by linking questions to sentences in a corpus of text. Shen and Lapata (2007) perform  
 6987 FrameNet semantic role labeling on the query, and then construct a weighted **bipartite**  
 6988 **graph**<sup>15</sup> between FrameNet semantic roles and the words and phrases in the sentence.  
 6989 This is done by first scoring all pairs of semantic roles and assignments, as shown in the  
 6990 top half of Figure 13.6. They then find the bipartite edge cover, which is the minimum  
 6991 weighted subset of edges such that each vertex has at least one edge, as shown in the  
 6992 bottom half of Figure 13.6. After analyzing the question in this manner, Shen and Lapata  
 6993 then find semantically-compatible sentences in the corpus, by performing graph matching  
 6994 on the bipartite graphs for the question and candidate answer sentences. Finally, the  
 6995 *expected answer phrase* in the question — typically the *wh*-word — is linked to a phrase in  
 6996 the candidate answer source, and that phrase is returned as the answer.

6997 **Relation extraction** The task of **relation extraction** involves identifying pairs of entities  
 6998 for which a given semantic relation holds (see § 17.2. For example, we might like to find  
 6999 all pairs  $(i, j)$  such that  $i$  is the INVENTOR-OF  $j$ . PropBank semantic role labeling can  
 7000 be applied to this task by identifying sentences whose verb signals the desired relation,  
 7001 and then extracting ARG1 and ARG2 as arguments. (To fully solve this task, these argu-  
 7002 ments must then be linked to entities, as described in chapter 17.) Christensen et al. (2010)  
 7003 compare a semantic role labeling system against a simpler approach based on surface pat-  
 7004 terns (Banko et al., 2007). They find that the SRL system is considerably more accurate,  
 7005 but that it is several orders of magnitude slower. Conversely, Barnickel et al. (2009) apply  
 7006 SENNA, a convolutional neural network SRL system (Collobert and Weston, 2008) to the  
 7007 task of identifying biomedical relations (e.g., which genes inhibit or activate each other).

<sup>15</sup>A 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.

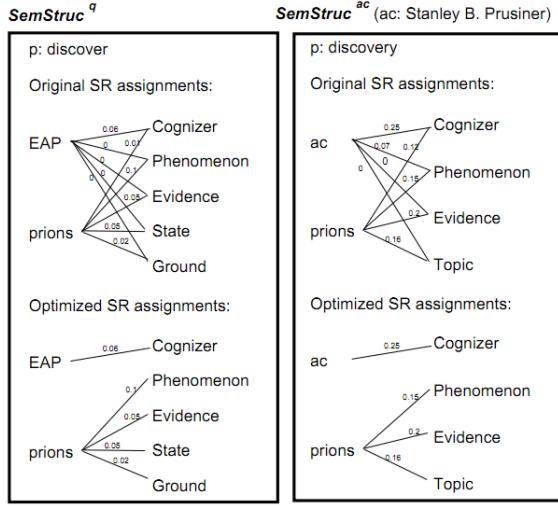


Figure 13.6: FrameNet semantic role labeling is used in factoid question answering, by aligning the semantic roles in the question (q) against those of sentences containing answer candidates (ac). “EAP” is the expected answer phrase, replacing the word *who* in the question. Figure reprinted from Shen and Lapata (2007) [todo: permission]

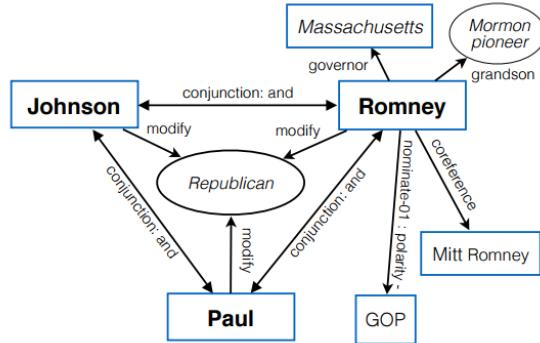


Figure 13.7: Fragment of AMR knowledge network for entity linking. Figure reprinted from Pan et al. (2015) [todo: permission]

7008 In comparison with a strong baseline that applies a set of rules to syntactic dependency  
 7009 structures (Fundel et al., 2007), the SRL system is faster but less accurate. One possible  
 7010 explanation for these divergent results is that Fundel et al. compare against a baseline  
 7011 which is carefully tuned for performance in a relatively narrow domain, while the system  
 7012 of Banko et al. is designed to analyze text across the entire web.

7013 **Entity linking** Another core task in information extraction is to link mentions of entities  
7014 (e.g., *Republican candidates like Romney, Paul, and Johnson* ...) to entities in a knowledge  
7015 base (e.g., LYNDON JOHNSON or GARY JOHNSON). This task, which is described in § 17.1,  
7016 is often performed by examining nearby “collaborator” mentions — in this case, *Romney*  
7017 and *Paul*. By jointly linking all such mentions, it is possible to arrive at a good overall  
7018 solution. Pan et al. (2015) apply AMR to this problem. For each entity, they construct a  
7019 knowledge network based on its semantic relations with other mentions within the same  
7020 sentence. They then rerank a set of candidate entities, based on the overlap between  
7021 the entity’s knowledge network and the semantic relations present in the sentence (Figure  
7022 13.7).

7023 **Exercises**

- 7024 1. Write out an event semantic representation for the following sentences. You may  
7025 make up your own predicates.
    - 7026 (13.26) *Abigail shares with Max.*
    - 7027 (13.27) *Abigail reluctantly shares a toy with Max.*
    - 7028 (13.28) *Abigail hates to share with Max.*
  - 7029 2. Find the PropBank framesets for *share* and *hate* at <http://verbs.colorado.edu/propbank/framesets-english-aliases/>, and rewrite your answers from the  
7030 previous question, using the thematic roles ARG0, ARG1, and ARG2.
  - 7032 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  
7033 sure about the parse, you can try an online parser such as <http://nlp.stanford.edu:8080/parser/>.  
7034
  - 7036 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  
7037 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*  
7039 and *Max* is OBL according to the Universal Dependency treebank (version 2).  
7040
  - 7041 5. PropBank semantic role labeling includes **reference arguments**, such as,
- 7042 (13.29) [AM-LOC The bed] on [R-AM-LOC which] I slept broke.<sup>16</sup>

---

<sup>16</sup>Example from 2013 NAACL tutorial slides by Shumin Wu

7043       The label R-AM-LOC indicates that word *which* is a reference to *The bed*, which ex-  
 7044       presses the location of the event. Reference arguments must have referents: the tag  
 7045       R-AM-LOC can appear only when AM-LOC also appears in the sentence. Show how  
 7046       to express this as a linear constraint, specifically for the tag R-AM-LOC. Be sure to  
 7047       correctly handle the case in which neither AM-LOC nor R-AM-LOC appear in the  
 7048       sentence.

- 7049     6. Explain how to express the constraints on semantic role labeling in Equation 13.8  
 7050       and Equation 13.9 in the general form  $Az \geq b$ .
- 7051     7. Download the FrameNet sample data (<https://framenet.icsi.berkeley.edu/fndrupal/fulltextIndex>), and train a bag-of-words classifier to predict the  
 7052       frame that is evoked by each verb in each example. Your classifier should build  
 7053       a bag-of-words from the sentence in which the frame-evoking lexical unit appears.  
 7054       [**todo: Somehow limit to one or a few lexical units.**] [**todo: use NLTK if possible**]
- 7055     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  
 7056       LSTM to predict tags. You will have to convert the downloaded instances to a BIO  
 7057       sequence labeling representation first.
- 7058     9. Produce the AMR annotations for the following examples:

- 7061       (13.30) The girl likes the boy.
- 7062       (13.31) The girl was liked by the boy.
- 7063       (13.32) Abigail likes Maxwell Aristotle.
- 7064       (13.33) The spy likes the attractive boy.
- 7065       (13.34) The girl doesn't like the boy.
- 7066       (13.35) The girl likes her dog.

7067       For (13.32), recall that multi-token names are created using op1, op2, etc. You will  
 7068       need to consult Banerescu et al. (2013) for (13.34), and Schneider et al. (2015) for  
 7069       (13.35). You may assume that *her* refers to *the girl* in this example.

- 7070     10. Using an off-the-shelf PropBank SRL system,<sup>17</sup> build a simplified question answer-  
 7071       ing system in the style of Shen and Lapata (2007). Specifically, your system should  
 7072       do the following:

---

<sup>17</sup>At 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](https://cogcomp.cs.illinois.edu/page/software_view/SRL)), and mate-tools (<https://code.google.com/archive/p/mate-tools/>).

- For each document in a collection, it should apply the semantic role labeler, and should store the output as a tuple.
- For a question, your system should again apply the semantic role labeler. If any of the roles are filled by a *wh*-pronoun, you should mark that role as the expected answer phrase (EAP).
- To answer the question, search for a stored tuple which matches the question as well as possible (same predicate, no incompatible semantic roles, and as many matching roles as possible). Align the EAP against its role filler in the stored tuple, and return this as the answer.

To evaluate your system, download a set of three news articles on the same topic, and write down five factoid questions that should be answerable from the articles. See if your system can answer these questions correctly. (If this problem is assigned to an entire class, you can build a large-scale test set and compare various approaches.)



7087 **Chapter 14**

7088 **Distributional and distributed  
7089 semantics**

7090 A recurring theme in natural language processing is the complexity of the mapping from  
7091 words to meaning. In chapter 4, we saw that a single word form, like *bank*, can have mul-  
7092 tiple meanings; conversely, a single meaning may be created by multiple surface forms,  
7093 a lexical semantic relationship known as **synonymy**. Despite this complex mapping be-  
7094 tween words and meaning, natural language processing systems usually rely on words  
7095 as the basic unit of analysis. This is especially true in semantics: the logical and frame  
7096 semantic methods from the previous two chapters rely on hand-crafted lexicons that map  
7097 from words to semantic predicates. But how can we analyze texts that contain words  
7098 that we haven't seen before? This chapter describes methods that learn representations  
7099 of word meaning by analyzing unlabeled data, vastly improving the generalizability of  
7100 natural language processing systems. The theory that makes it possible to acquire mean-  
7101 ingful representations from unlabeled data is the **distributional hypothesis**.

7102 **14.1 The distributional hypothesis**

7103 Here's a word you may not know: *tezgüino* (the example is from Lin, 1998). If you do not  
7104 know the meaning of *tezgüino*, then you are in the same situation as a natural language  
7105 processing system when it encounters a word that did not appear in its training data.  
7106 Now suppose you see that *tezgüino* is used in the following contexts:

- 7107 (14.1) A bottle of \_\_\_\_\_ is on the table.
- 7108 (14.2) Everybody likes \_\_\_\_\_.
- 7109 (14.3) Don't have \_\_\_\_\_ before you drive.
- 7110 (14.4) We make \_\_\_\_\_ out of corn.

	(14.1)	(14.2)	(14.3)	(14.4)	...
<i>tezgüino</i>	1	1	1	1	
<i>loud</i>	0	0	0	0	
<i>motor oil</i>	1	0	0	1	
<i>tortillas</i>	0	1	0	1	
<i>choices</i>	0	1	0	0	
<i>wine</i>	1	1	1	0	

Table 14.1: Distributional statistics for *tezgüino* and five related terms

What other words fit into these contexts? How about: *loud*, *motor oil*, *tortillas*, *choices*, *wine*? Each row of Table 14.1 is a vector that summarizes the contextual properties for each word, with a value of one for contexts in which the word can appear, and a value of zero for contexts in which it cannot. Based on these vectors, we can conclude: *wine* is very similar to *tezgüino*; *motor oil* and *tortillas* are fairly similar to *tezgüino*; *loud* is completely different.

These vectors, which we will call **word representations**, describe the **distributional** properties of each word. Does vector similarity imply semantic similarity? This is the **distributional hypothesis**, stated by Firth (1957) as: “You shall know a word by the company it keeps.” The distributional hypothesis has stood the test of time: distributional statistics are a core part of language technology today, because they make it possible to leverage large amounts of unlabeled data to learn about rare words that do not appear in labeled training data.

Distributional statistics have a striking ability to capture lexical semantic relationships such as analogies. Figure 14.1 shows two examples, based on two-dimensional projections of distributional **word embeddings**, discussed later in this chapter. In each case, word-pair relationships correspond to regular linear patterns in this two dimensional space. No labeled data about the nature of these relationships was required to identify this underlying structure.

**Distributional** semantics are computed from context statistics. **Distributed** semantics are a related but distinct idea: that meaning can be represented by numerical vectors rather than symbolic structures. Distributed representations are often estimated from distributional statistics, as in latent semantic analysis and WORD2VEC, described later in this chapter. However, distributed representations can also be learned in a supervised fashion from labeled data, as in the neural classification models encountered in chapter 3.

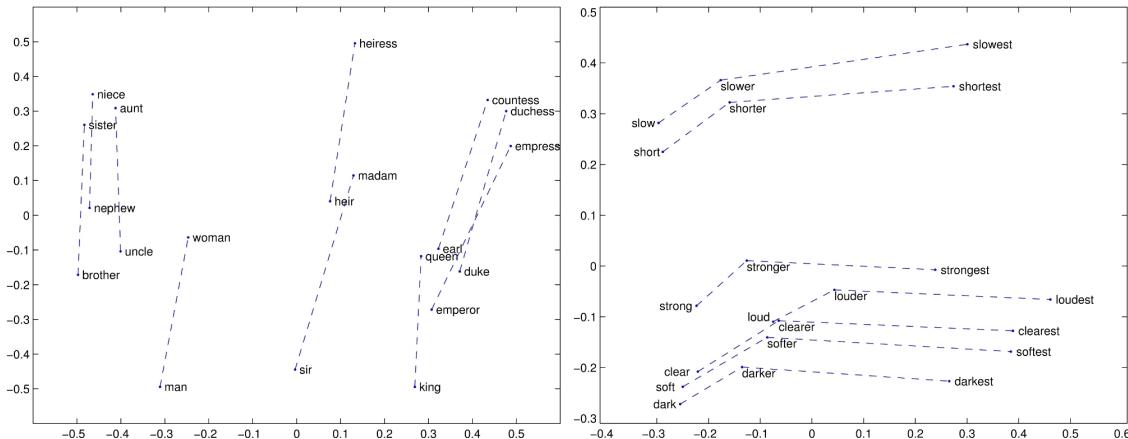


Figure 14.1: Lexical semantic relationships have regular linear structures in two dimensional projections of distributional statistics. From [http://nlp.stanford.edu/projects/glove/.\[todo: redo to make words bigger?\]](http://nlp.stanford.edu/projects/glove/.[todo: redo to make words bigger?])

## 7136 14.2 Design decisions for word representations

7137 There are many approaches for computing word representations, but most can be distin-  
 7138 guished on three main dimensions: the nature of the representation, the source of context-  
 7139 ual information, and the estimation procedure.

### 7140 14.2.1 Representation

7141 Today, the dominant word representations are  $k$ -dimensional vectors of real numbers,  
 7142 known as **word embeddings**. (The name is due to the fact that each discrete word is em-  
 7143 bedded in a continuous vector space.) This representation dates back at least to the late  
 7144 1980s (Deerwester et al., 1990), and is used in popular techniques such as WORD2VEC (Mikolov  
 7145 et al., 2013).

7146 Word embeddings are well suited for neural networks, where they can be plugged  
 7147 in as inputs. They can also be applied in linear classifiers and structure prediction mod-  
 7148 els (Turian et al., 2010), although it can be difficult to learn linear models that employ  
 7149 real-valued features (Kummerfeld et al., 2015). A popular alternative is bit-string rep-  
 7150 resentations, such as **Brown clusters** (§ 14.4), in which each word is represented by a  
 7151 variable-length sequence of zeros and ones (Brown et al., 1992).

7152 Another representational question is whether to estimate one embedding per surface  
 7153 form (e.g., *bank*), or to estimate distinct embeddings for each word sense or synset. In-  
 7154 tuitively, if word representations are to capture the meaning of individual words, then  
 7155 words with multiple meanings should have multiple embeddings. This can be achieved

---

*The moment one learns English, complications set in* (Alfau, 1999)

---

Brown Clusters (Brown et al., 1992)	{one}
WORD2VEC (Mikolov et al., 2013) ( $h = 2$ )	{moment, one, English, complications}
Structured WORD2VEC (Ling et al., 2015) ( $h = 2$ )	$\{(moment, -2), (one, -1), (English, +1), (complications, +2)\}$
Dependency contexts (Levy and Goldberg, 2014)	$\{(one, \text{NSUBJ}), (\text{English}, \text{DOBJ}), (moment, \text{ACL}^{-1})\}$

---

Table 14.2: Contexts for the word *learns*, according to various word representations. For dependency context, *(one, NSUBJ)* means that there is a relation of type NSUBJ (nominal subject) **to** the word *one*, and *(moment, ACL<sup>-1</sup>)* means that there is a relation of type ACL (adjectival clause) **from** the word *moment*.

7156 by integrating unsupervised clustering with word embedding estimation (Huang and  
 7157 Yates, 2012; Li and Jurafsky, 2015). However, Arora et al. (2016) argue that it is unnec-  
 7158 essary to model distinct word senses explicitly, because the embeddings for each surface  
 7159 form are a linear combination of the embeddings of the underlying senses.

### 7160 14.2.2 Context

7161 The distributional hypothesis says that word meaning is related to the “contexts” in which  
 7162 the word appears, but context can be defined in many ways. In the *tezgiiino* example, con-  
 7163 texts are entire sentences, but in practice there are far too many sentences. At the oppo-  
 7164 site extreme, the context could be defined as the immediately preceding word; this is the  
 7165 context considered in Brown clusters. WORD2VEC takes an intermediate approach, using  
 7166 local neighborhoods of words (e.g.,  $h = 5$ ) as contexts (Mikolov et al., 2013). Contexts  
 7167 can also be much larger: for example, in **latent semantic analysis**, each word’s context  
 7168 vector includes an entry per document, with a value of one if the word appears in the  
 7169 document (Deerwester et al., 1990); in **explicit semantic analysis**, these documents are  
 7170 Wikipedia pages (Gabrilovich and Markovitch, 2007).

7171 Words in context can be labeled by their position with respect to the target word  $w_m$   
 7172 (e.g., two words before, one word after), which makes the resulting word representations  
 7173 more sensitive to syntactic differences (Ling et al., 2015). Another way to incorporate  
 7174 syntax is to perform parsing as a preprocessing step, and then form context vectors from  
 7175 the dependency edges (Levy and Goldberg, 2014) or predicate-argument relations (Lin,  
 7176 1998). The resulting context vectors for several of these methods are shown in Table 14.2.

7177 The choice of context has a profound effect on the resulting representations, which

7178 can be viewed in terms of word similarity. Applying latent semantic analysis (§ 14.3) to  
 7179 contexts of size  $h = 2$  and  $h = 30$  yields the following nearest-neighbors for the word  
 7180 *dog*:<sup>1</sup>

- 7181 • ( $h = 2$ ): *cat, horse, fox, pet, rabbit, pig, animal, mongrel, sheep, pigeon*  
 7182 • ( $h = 30$ ): *kennel, puppy, pet, bitch, terrier, rottweiler, canine, cat, to bark, Alsatian*

7183 Which word list is better? Each word in the  $h = 2$  list is an animal, reflecting the fact that  
 7184 locally, the word *dog* tends to appear in the same contexts as other animal types (e.g., *pet*  
 7185 *the dog, feed the dog*). In the  $h = 30$  list, nearly everything is dog-related, including specific  
 7186 breeds such as *rottweiler* and *Alsatian*. The list also includes words that are not animals  
 7187 (*kennel*), and in one case (*to bark*), is not a noun at all. The 2-word context window is more  
 7188 sensitive to syntax, while the 30-word window is more sensitive to topic.

### 7189 14.2.3 Estimation

7190 Word embeddings are estimated by optimizing some objective: the likelihood of a set of  
 7191 unlabeled data (or a closely related quantity), or the reconstruction of a matrix of context  
 7192 counts, similar to Table 14.1.

7193 **Maximum likelihood estimation** Likelihood-based optimization is derived from the  
 7194 objective  $\log p(\mathbf{w}; \mathbf{U})$ , where  $\mathbf{U} \in \mathbb{R}^{K \times V}$  is matrix of word embeddings, and  $\mathbf{w} =$   
 7195  $\{\mathbf{w}_m\}_{m=1}^M$  is a corpus, represented as a list of  $M$  tokens. Recurrent neural network lan-  
 7196 guage models (§ 6.3) optimize this objective directly, backpropagating to the input word  
 7197 embeddings through the recurrent structure. However, state-of-the-art word embeddings  
 7198 employ huge corpora with hundreds of billions of tokens, and recurrent architectures are  
 7199 difficult to scale to such data. As a result, likelihood-based word embeddings are usually  
 7200 based on simplified likelihoods or heuristic approximations.

**Matrix factorization** The matrix  $\mathbf{C} = \{\text{count}(i, j)\}$  stores the co-occurrence counts of  
 word  $i$  and context  $j$ . Word representations can be obtained by approximately factoring  
 this matrix, so that  $\text{count}(i, j)$  is approximated by a function of a word embedding  $\mathbf{u}_i$  and  
 a context embedding  $\mathbf{v}_j$ . These embeddings can be obtained by minimizing the norm of  
 the reconstruction error,

$$\min_{\mathbf{u}, \mathbf{v}} \|\mathbf{C} - \tilde{\mathbf{C}}(\mathbf{u}, \mathbf{v})\|_F, \quad [14.1]$$

---

<sup>1</sup>The 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/>

7201 where  $\tilde{\mathbf{C}}(\mathbf{u}, \mathbf{v})$  is the approximate reconstruction resulting from the embeddings  $\mathbf{u}$  and  
 7202  $\mathbf{v}$ , and  $\|\mathbf{X}\|_F$  indicates the Frobenius norm,  $\sum_{i,j} x_{i,j}^2$ . Rather than factoring the matrix of  
 7203 word-context counts directly, it is often helpful to transform these counts using information-  
 7204 theoretic metrics such as **pointwise mutual information** (PMI), described in the next sec-  
 7205 tion.

### 7206 14.3 Latent semantic analysis

Latent semantic analysis (LSA) is one of the oldest approaches to distributed semantics (Deerwester et al., 1990). It induces continuous vector representations of words by factoring a matrix of word and context counts, using **truncated singular value decomposition** (SVD),

$$\min_{\mathbf{U} \in \mathbb{R}^{V \times K}, \mathbf{S} \in \mathbb{R}^{K \times K}, \mathbf{V} \in \mathbb{R}^{|\mathcal{C}| \times K}} \|\mathbf{C} - \mathbf{USV}^\top\|_F \quad [14.2]$$

$$\text{s.t. } \mathbf{U}^\top \mathbf{U} = \mathbb{I} \quad [14.3]$$

$$\mathbf{V}^\top \mathbf{V} = \mathbb{I} \quad [14.4]$$

$$\forall i \neq j, \mathbf{S}_{i,j} = 0, \quad [14.5]$$

7207 where  $V$  is the size of the vocabulary,  $|\mathcal{C}|$  is the number of contexts, and  $K$  is size of the  
 7208 resulting embeddings, which are set equal to the rows of the matrix  $\mathbf{U}$ . The matrix  $\mathbf{S}$  is  
 7209 constrained to be diagonal (these diagonal elements are called the singular values), and  
 7210 the columns of the product  $\mathbf{SV}^\top$  provide descriptions of the contexts. Each element  $c_{i,j}$  is  
 7211 then reconstructed as a **bilinear product**,

$$c_{i,j} \approx \sum_{k=1}^K u_{i,k} s_k v_{j,k}. \quad [14.6]$$

7212 The objective is to minimize the sum of squared approximation errors. The orthonormality  
 7213 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  
 7214 uncorrelated, so that each dimension conveys unique information. Efficient implemen-  
 7215 tations of truncated singular value decomposition are available in numerical computing  
 7216 packages such as `scipy` and `matlab`.<sup>2</sup>

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

---

<sup>2</sup>An important implementation detail is to represent  $\mathbf{C}$  as a **sparse matrix**, so that the storage cost is equal to the number of non-zero entries, rather than the size  $V \times |\mathcal{C}|$ .

context  $j$ ,

$$\text{PMI}(i, j) = \log \frac{p(i, j)}{p(i)p(j)} = \log \frac{p(i | j)p(j)}{p(i)p(j)} = \log \frac{p(i | j)}{p(i)} \quad [14.7]$$

$$= \log \text{count}(i, j) - \log \sum_{i'=1}^V \text{count}(i', j) \quad [14.8]$$

$$- \log \sum_{j' \in \mathcal{C}} \text{count}(i, j') + \log \sum_{i'=1}^V \sum_{j' \in \mathcal{C}} \text{count}(i', j'). \quad [14.9]$$

7217 The pointwise mutual information can be viewed as the logarithm of the ratio of the con-  
 7218 ditional probability of word  $i$  in context  $j$  to the marginal probability of word  $i$  in all  
 7219 contexts. When word  $i$  is statistically associated with context  $j$ , the ratio will be greater  
 7220 than one, so  $\text{PMI}(i, j) > 0$ . The PMI transformation focuses latent semantic analysis on re-  
 7221 constructing strong word-context associations, rather than on reconstructing large counts.

7222 The PMI is negative when a word and context occur together less often than if they  
 7223 were independent, but such negative correlations are unreliable because counts of rare  
 7224 events have high variance. Furthermore, the PMI is undefined when  $\text{count}(i, j) = 0$ . One  
 7225 solution to these problems is to use the **Positive PMI** (PPMI),

$$\text{PPMI}(i, j) = \begin{cases} \text{PMI}(i, j), & p(i | j) > p(i) \\ 0, & \text{otherwise.} \end{cases} \quad [14.10]$$

7226 Bullinaria and Levy (2007) compare a range of matrix transformations for latent se-  
 7227 mantic analysis, using a battery of tasks related to word meaning and word similarity  
 7228 (for more on evaluation, see § 14.6). They find that PPMI-based latent semantic analysis  
 7229 yields strong performance on a battery of tasks related to word meaning: for example,  
 7230 PPMI-based LSA vectors can be used to solve multiple-choice word similarity questions  
 7231 from the Test of English as a Foreign Language (TOEFL), obtaining 85% accuracy.

## 7232 14.4 Brown clusters

7233 Learning algorithms like perceptron and conditional random fields often perform better  
 7234 with discrete feature vectors. A simple way to obtain discrete representations from distri-  
 7235 butional statistics is by clustering (§ 5.1.1), so that words in the same cluster have similar  
 7236 distributional statistics. This can help in downstream tasks, by sharing features between  
 7237 all words in the same cluster. However, there is an obvious tradeoff: if the number of clus-  
 7238 ters is too small, the words in each cluster will not have much in common; if the number  
 7239 of clusters is too large, then the learner will not see enough examples from each cluster to  
 7240 generalize.

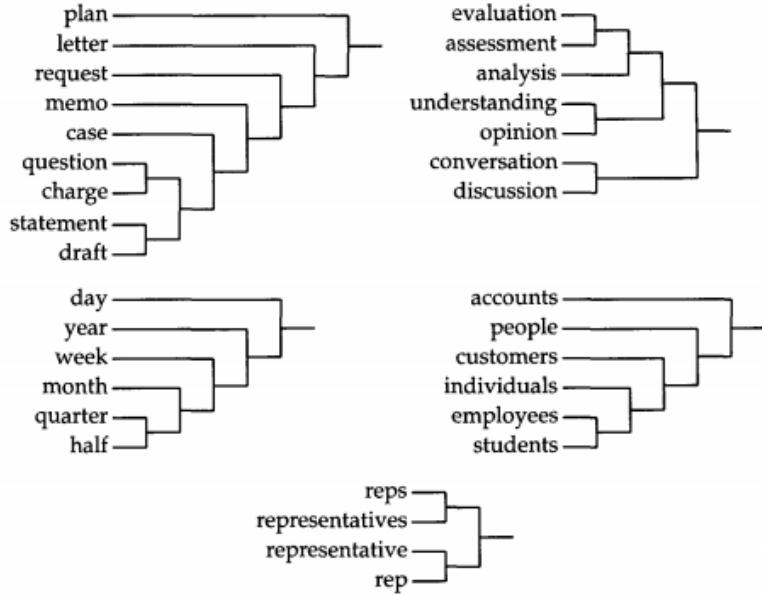


Figure 14.2: Some subtrees produced by bottom-up Brown clustering (Miller et al., 2004) on news text [todo: permission]

7241 A solution to this problem is **hierarchical clustering**: using the distributional statistics  
 7242 to induce a tree-structured representation. Fragments of **Brown cluster** trees are shown in  
 7243 Figure 14.2 and Table 14.3. Each word’s representation consists of a binary string describ-  
 7244 ing a path through the tree: 0 for taking the left branch, and 1 for taking the right branch.  
 7245 In the subtree in the upper right of the figure, the representation of the word *conversation*  
 7246 is 10; the representation of the word *assessment* is 0001. Bitstring prefixes capture simila-  
 7247 rity at varying levels of specificity, and it is common to use the first eight, twelve, sixteen,  
 7248 and twenty bits as features in tasks such as named entity recognition (Miller et al., 2004)  
 7249 and dependency parsing (Koo et al., 2008).

Hierarchical trees can be induced from a likelihood-based objective, using a discrete latent variable  $k_i \in \{1, 2, \dots, K\}$  to represent the cluster of word  $i$ :

$$\log p(\mathbf{w}; \mathbf{k}) \approx \sum_{m=1}^M \log p(w_m | w_{m-1}; \mathbf{k}) \quad [14.11]$$

$$\triangleq \sum_{m=1}^M \log p(w_m | k_{w_m}) + \log p(k_{w_m} | k_{w_{m-1}}). \quad [14.12]$$

7250 This is similar to a hidden Markov model, with the crucial difference that each word can

bitstring	ten most frequent words
01111010 <b>0111</b>	<i>excited thankful grateful stoked pumped anxious hyped psyched exited geeked</i>
01111010 <b>100</b>	<i>talking talkin complaining talkn bitching tlkn tlkin bragging rav- ing +k</i>
01111010 <b>1010</b>	<i>thinking thinkin dreaming worrying thinkn speakin reminiscing dreamin daydreaming fantasizing</i>
01111010 <b>1011</b>	<i>saying sayin suggesting stating sayn jokin talmbout implying insisting 5'2</i>
01111010 <b>1100</b>	<i>wonder dunno wondered duno donno dno doно wonda wounder dunnoe</i>
01111010 <b>1101</b>	<i>wondering wonders debating deciding pondering unsure won- derin debatin woundering wondern</i>
01111010 <b>1110</b>	<i>sure suree suuure suure sure- surre sures shuree</i>

Table 14.3: Fragment of a Brown clustering of Twitter data (Owoputi et al., 2013). Each row is a leaf in the tree, showing the ten most frequent words. This part of the tree emphasizes verbs of communicating and knowing, especially in the present participle. Each leaf node includes orthographic variants (*thinking*, *thinkin*, *thinkn*), semantically related terms (*excited*, *thankful*, *grateful*), and some outliers (*5'2*, *+k*). See [http://www.cs.cmu.edu/~ark/TweetNLP/cluster\\_viewer.html](http://www.cs.cmu.edu/~ark/TweetNLP/cluster_viewer.html) for more.

7251 be emitted from only a single cluster:  $\forall k \neq k_{w_m}, p(w_m | k) = 0$ .

Using the objective in Equation 14.12, the Brown clustering tree can be constructed from the bottom up: begin with each word in its own cluster, and incrementally merge clusters until only a single cluster remains. At each step, we merge the pair of clusters such that the objective in Equation 14.12 is maximized. Although the objective seems to involve a sum over the entire corpus, the score for each merger can be computed from the cluster-to-cluster co-occurrence counts. These counts can be updated incrementally as the clustering proceeds. The optimal merge at each step can be shown to maximize the **average mutual information**,

$$I(\mathbf{k}) = \sum_{k_1=1}^K \sum_{k_2=1}^K p(k_1, k_2) \times \text{PMI}(k_1, k_2) \quad [14.13]$$

$$p(k_1, k_2) = \frac{\text{count}(k_1, k_2)}{\sum_{k_{1'}=1}^K \sum_{k_{2'}=1}^K \text{count}(k_{1'}, k_{2'})},$$

7252 where  $p(k_1, k_2)$  is the joint probability of a bigram involving a word in cluster  $k_1$  followed  
7253 by a word in  $k_2$ . This probability and the PMI are both computed from the co-occurrence

---

**Algorithm 17** Exchange clustering algorithm. Assumes that words are sorted by frequency, and that MAXMI finds the cluster pair whose merger maximizes the mutual information, as defined in Equation 14.13.

---

```

procedure EXCHANGECLUSTERING({count( $\cdot, \cdot$ )},  $K$ )
    for  $i \in 1 \dots K$  do                                 $\triangleright$  Initialization
         $k_i \leftarrow i$ ,  $i = 1, 2, \dots, K$ 
        for  $j \in 1 \dots K$  do
             $c_{i,j} \leftarrow \text{count}(i, j)$ 
         $\tau \leftarrow \{(i)\}_{i=1}^K$ 
        for  $i \in \{K + 1, K + 2, \dots, V\}$  do       $\triangleright$  Iteratively add each word to the clustering
             $\tau \leftarrow \tau \cup (i)$ 
            for  $k \in \tau$  do
                 $c_{k,i} \leftarrow \text{count}(k, i)$ 
                 $c_{i,k} \leftarrow \text{count}(i, k)$ 
             $\hat{i}, \hat{j} \leftarrow \text{MAXMI}(\mathbf{C})$ 
             $\tau, \mathbf{C} \leftarrow \text{MERGE}(\hat{i}, \hat{j}, \mathbf{C}, \tau)$ 
        repeat                                          $\triangleright$  Merge the remaining clusters into a tree
             $\hat{i}, \hat{j} \leftarrow \text{MAXMI}(\mathbf{C}, \tau)$ 
             $\tau, \mathbf{C} \leftarrow \text{MERGE}(\hat{i}, \hat{j}, \mathbf{C}, \tau)$ 
        until  $|\tau| = 1$ 
        return  $\tau$ 
procedure MERGE( $i, j, \mathbf{C}, \tau$ )
     $\tau \leftarrow \tau \setminus i \setminus j \cup (i, j)$            $\triangleright$  Merge the clusters in the tree
    for  $k \in \tau$  do                                 $\triangleright$  Aggregate the counts across the merged clusters
         $c_{k,(i,j)} \leftarrow c_{k,i} + c_{k,j}$ 
         $c_{(i,j),k} \leftarrow c_{i,k} + c_{j,k}$ 
    return  $\tau, \mathbf{C}$ 

```

---

7254 counts between clusters. After each merger, the co-occurrence vectors for the merged  
 7255 clusters are simply added up, so that the next optimal merger can be found efficiently.

7256 This bottom-up procedure requires iterating over the entire vocabulary, and evaluating  
 7257  $K_t^2$  possible mergers at each step, where  $K_t$  is the current number of clusters at step  $t$   
 7258 of the algorithm. Furthermore, computing the score for each merger involves a sum over  
 7259  $K_t^2$  clusters. The maximum number of clusters is  $K_0 = V$ , which occurs when every word  
 7260 is in its own cluster at the beginning of the algorithm. The time complexity is thus  $\mathcal{O}(V^5)$ .

7261 To avoid this complexity, practical implementations use a heuristic approximation  
 7262 called **exchange clustering**. The  $K$  most common words are placed in clusters of their  
 7263 own at the beginning of the process. We then consider the next most common word, and

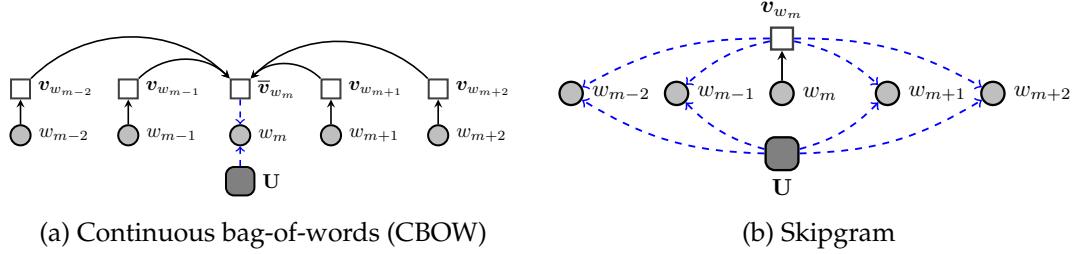


Figure 14.3: The CBOW and skipgram variants of WORD2VEC. The parameter  $\mathbf{U}$  is the matrix of word embeddings, and each  $\mathbf{v}_m$  is the context embedding for word  $w_m$ .

7264 merge it with one of the existing clusters. This continues until the entire vocabulary has  
 7265 been incorporated, at which point the  $K$  clusters are merged down to a single cluster,  
 7266 forming a tree. The algorithm never considers more than  $K + 1$  clusters at any step, and  
 7267 the complexity is  $\mathcal{O}(VK + V \log V)$ , with the second term representing the cost of sorting  
 7268 the words at the beginning of the algorithm.

## 7269 14.5 Neural word embeddings

7270 Neural word embeddings combine aspects of the previous two methods: like latent se-  
 7271 mantic analysis, they are a continuous vector representation; like Brown clusters, they are  
 7272 trained from a likelihood-based objective. Let the vector  $\mathbf{u}_i$  represent the  $K$ -dimensional  
**7273 embedding** for word  $i$ , and let  $\mathbf{v}_j$  represent the  $K$ -dimensional embedding for context  
 7274  $j$ . The inner product  $\mathbf{u}_i \cdot \mathbf{v}_j$  represents the compatibility between word  $i$  and context  $j$ .  
 7275 By incorporating this inner product into an approximation to the log-likelihood of a cor-  
 7276 pus, it is possible to estimate both parameters by backpropagation. WORD2VEC (Mikolov  
 7277 et al., 2013) includes two such approximations: continuous bag-of-words (CBOW) and  
 7278 skipgrams.

### 7279 14.5.1 Continuous bag-of-words (CBOW)

7280 In recurrent neural network language models, each word  $w_m$  is conditioned on a recurrently-  
 7281 updated state vector, which is based on word representations going all the way back to the  
 7282 beginning of the text. The **continuous bag-of-words (CBOW)** model is a simplification:  
 7283 the local context is computed as an average of embeddings for words in the immediate  
 7284 neighborhood  $m - h, m - h + 1, \dots, m + h - 1, m + h$ ,

$$\bar{\mathbf{v}}_m = \frac{1}{2h} \sum_{n=1}^h \mathbf{v}_{w_{m+n}} + \mathbf{v}_{w_{m-n}}. \quad [14.14]$$

7285 Thus, CBOW is a bag-of-words model, because the order of the context words does not  
 7286 matter; it is continuous, because rather than conditioning on the words themselves, we  
 7287 condition on a continuous vector constructed from the word embeddings. The parameter  
 7288  $h$  determines the neighborhood size, which Mikolov et al. (2013) set to  $h = 4$ .

The CBOW model optimizes an approximation to the corpus log-likelihood,

$$\log p(\mathbf{w}) \approx \sum_{m=1}^M \log p(w_m | w_{m-h}, w_{m-h+1}, \dots, w_{m+h-1}, w_{m+h}) \quad [14.15]$$

$$= \sum_{m=1}^M \log \frac{\exp(\mathbf{u}_{w_m} \cdot \bar{\mathbf{v}}_m)}{\sum_{j=1}^V \exp(\mathbf{u}_j \cdot \bar{\mathbf{v}}_m)} \quad [14.16]$$

$$= \sum_{m=1}^M \mathbf{u}_{w_m} \cdot \bar{\mathbf{v}}_m - \log \sum_{j=1}^V \exp(\mathbf{u}_j \cdot \bar{\mathbf{v}}_m). \quad [14.17]$$

### 7289 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:

$$\log p(\mathbf{w}) \approx \sum_{m=1}^M \sum_{n=1}^{h_m} \log p(w_{m-n} | w_m) + \log p(w_{m+n} | w_m) \quad [14.18]$$

$$= \sum_{m=1}^M \sum_{n=1}^{h_m} \log \frac{\exp(\mathbf{u}_{w_{m-n}} \cdot \mathbf{v}_{w_m})}{\sum_{j=1}^V \exp(\mathbf{u}_j \cdot \mathbf{v}_{w_m})} + \log \frac{\exp(\mathbf{u}_{w_{m+n}} \cdot \mathbf{v}_{w_m})}{\sum_{j=1}^V \exp(\mathbf{u}_j \cdot \mathbf{v}_{w_m})} \quad [14.19]$$

$$= \sum_{m=1}^M \sum_{n=1}^{h_m} \mathbf{u}_{w_{m-n}} \cdot \mathbf{v}_{w_m} + \mathbf{u}_{w_{m+n}} \cdot \mathbf{v}_{w_m} - 2 \log \sum_{j=1}^V \exp(\mathbf{u}_j \cdot \mathbf{v}_{w_m}). \quad [14.20]$$

7290 In the skipgram approximation, each word is generated multiple times; each time it is con-  
 7291 ditioned only on a single word. This makes it possible to avoid averaging the word vec-  
 7292 tors, as in the CBOW model. The local neighborhood size  $h_m$  is randomly sampled from  
 7293 a uniform categorical distribution over the range  $\{1, 2, \dots, h_{\max}\}$ ; Mikolov et al. (2013) set  
 7294  $h_{\max} = 10$ . Because the neighborhood grows outward with  $h$ , this approach has the effect  
 7295 of weighting near neighbors more than distant ones. Skipgram performs better on most  
 7296 evaluations than CBOW (see § 14.6 for details of how to evaluate word representations),  
 7297 but CBOW is faster to train (Mikolov et al., 2013).

### 7298 14.5.3 Computational complexity

7299 The WORD2VEC models can be viewed as an efficient alternative to recurrent neural net-  
 7300 work language models, which involve a recurrent state update whose time complexity

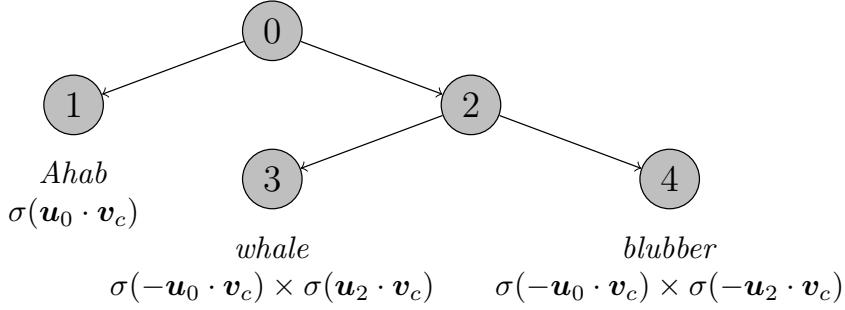


Figure 14.4: A fragment of a hierarchical softmax tree. The probability of each word is computed as a product of probabilities of local branching decisions in the tree.

is quadratic in the size of the recurrent state vector. CBOW and skipgram avoid this computation, and incur only a linear time complexity in the size of the word and context representations. However, all three models compute a normalized probability over word tokens; a naïve implementation of this probability requires summing over the entire vocabulary. The time complexity of this sum is  $\mathcal{O}(V \times K)$ , which dominates all other computational costs. There are two solutions: **hierarchical softmax**, a tree-based computation that reduces the cost to a logarithm of the size of the vocabulary; and **negative sampling**, an approximation that eliminates the dependence on vocabulary size. Both methods are also applicable to RNN language models.

#### 14.5.3.1 Hierarchical softmax

In Brown clustering, the vocabulary is organized into a binary tree. Mnih and Hinton (2008) show that the normalized probability over words in the vocabulary can be reparametrized as a probability over paths through such a tree. This **hierarchical softmax** probability is computed as a product of binary decisions over whether to move left or right through the tree, with each binary decision represented as a sigmoid function of the inner product between the context embedding  $\mathbf{v}_c$  and an output embedding associated with the node  $\mathbf{u}_n$ ,

$$\Pr(\text{left at } n \mid c) = \sigma(\mathbf{u}_n \cdot \mathbf{v}_c) \quad [14.21]$$

$$\Pr(\text{right at } n \mid c) = 1 - \sigma(\mathbf{u}_n \cdot \mathbf{v}_c) = \sigma(-\mathbf{u}_n \cdot \mathbf{v}_c), \quad [14.22]$$

where  $\sigma$  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

7316 logarithmic in the number of leaf nodes, and thus the number of multiplications is equal  
 7317 to  $\mathcal{O}(\log V)$ . The number of non-leaf nodes is equal to  $\mathcal{O}(2V - 1)$ , so the number of pa-  
 7318 rameters to be estimated increases by only a small multiple. The tree can be constructed  
 7319 using an incremental clustering procedure similar to hierarchical Brown clusters (Mnih  
 7320 and Hinton, 2008), or by using the Huffman (1952) encoding algorithm for lossless com-  
 7321 pression.

7322 **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**:

$$\psi(i, j) = \log \sigma(\mathbf{u}_i \cdot \mathbf{v}_j) + \sum_{i' \in \mathcal{W}_{\text{neg}}} \log(1 - \sigma(\mathbf{u}_{i'} \cdot \mathbf{v}_j)), \quad [14.23]$$

7323 where  $\psi(i, j)$  is the score for word  $i$  in context  $j$ , and  $\mathcal{W}_{\text{neg}}$  is the set of negative samples.  
 7324 The objective is to maximize the sum over the corpus,  $\sum_{m=1}^M \psi(w_m, c_m)$ , where  $w_m$  is  
 7325 token  $m$  and  $c_m$  is the associated context.

7326 The set of negative samples  $\mathcal{W}_{\text{neg}}$  is obtained by sampling from a unigram language  
 7327 model. Mikolov et al. (2013) construct this unigram language model by exponentiating  
 7328 the empirical word probabilities, setting  $\hat{p}(i) \propto (\text{count}(i))^{\frac{3}{4}}$ . This has the effect of redi-  
 7329 tributing probability mass from common to rare words. The number of negative samples  
 7330 increases the time complexity of training by a constant factor. Mikolov et al. (2013) report  
 7331 that 5-20 negative samples works for small training sets, and that two to five samples  
 7332 suffice for larger corpora.

7333 **14.5.4 Word embeddings as matrix factorization**

7334 The negative sampling objective in Equation 14.23 can be justified as an efficient approx-  
 7335 imation to the log-likelihood, but it is also closely linked to the matrix factorization ob-  
 7336 jective employed in latent semantic analysis. For a matrix of word-context pairs in which  
 7337 all counts are non-zero, negative sampling is equivalent to factorization of the matrix  $M$ ,  
 7338 where  $M_{ij} = \text{PMI}(i, j) - \log k$ : each cell in the matrix is equal to the pointwise mutual  
 7339 information of the word and context, shifted by  $\log k$ , with  $k$  equal to the number of neg-  
 7340 ative samples (Levy and Goldberg, 2014). For word-context pairs that are not observed in  
 7341 the data, the pointwise mutual information is  $-\infty$ , but this can be addressed by consid-  
 7342 ering only PMI values that are greater than  $\log k$ , resulting in a matrix of **shifted positive**

7343 **pointwise mutual information,**

$$M_{ij} = \max(0, \text{PMI}(i, j) - \log k). \quad [14.24]$$

7344 Word embeddings are obtained by factoring this matrix with truncated singular value  
7345 decomposition.

GloVe (“global vectors”) are a closely related approach (Pennington et al., 2014), in which the matrix to be factored is constructed from log co-occurrence counts,  $M_{ij} = \log \text{count}(i, j)$ . The word embeddings are estimated by minimizing the sum of squares,

$$\begin{aligned} \min_{\mathbf{u}, \mathbf{v}, b, \tilde{b}} \quad & \sum_{j=1}^V \sum_{j \in C} f(M_{ij}) \left( \widehat{\log M_{ij}} - \log M_{ij} \right)^2 \\ \text{s.t.} \quad & \widehat{\log M_{ij}} = \mathbf{u}_i \cdot \mathbf{v}_j + b_i + \tilde{b}_j, \end{aligned} \quad [14.25]$$

7346 where  $b_i$  and  $\tilde{b}_j$  are offsets for word  $i$  and context  $j$ , which are estimated jointly with the  
7347 embeddings  $\mathbf{u}$  and  $\mathbf{v}$ . The weighting function  $f(M_{ij})$  is set to be zero at  $M_{ij} = 0$ , thus  
7348 avoiding the problem of taking the logarithm of zero counts; it saturates at  $M_{ij} = m_{\max}$ ,  
7349 thus avoiding the problem of overcounting common word-context pairs. This heuristic  
7350 turns out to be critical to the method’s performance.

7351 The time complexity of sparse matrix reconstruction is determined by the number of  
7352 non-zero word-context counts. Pennington et al. (2014) show that this number grows  
7353 sublinearly with the size of the dataset: roughly  $\mathcal{O}(N^{0.8})$  for typical English corpora. In  
7354 contrast, the time complexity of WORD2VEC is linear in the corpus size. Computing the co-  
7355 occurrence counts also requires linear time in the size of the corpus, but this operation can  
7356 easily be parallelized using MapReduce-style algorithms (Dean and Ghemawat, 2008).

## 7357 14.6 Evaluating word embeddings

7358 Distributed word representations can be evaluated in two main ways. **Intrinsic** evalua-  
7359 tions test whether the representations cohere with our intuitions about word meaning.  
7360 **Extrinsic** evaluations test whether they are useful for downstream tasks, such as sequence  
7361 labeling.

### 7362 14.6.1 Intrinsic evaluations

7363 A basic question for word embeddings is whether the similarity of words  $i$  and  $j$  is re-  
7364 flected in the similarity of the vectors  $\mathbf{u}_i$  and  $\mathbf{u}_j$ . **Cosine similarity** is typically used to  
7365 compare two word embeddings,

$$\cos(\mathbf{u}_i, \mathbf{u}_j) = \frac{\mathbf{u}_i \cdot \mathbf{u}_j}{\|\mathbf{u}_i\|_2 \times \|\mathbf{u}_j\|_2}. \quad [14.26]$$

word 1	word 2	similarity
<i>love</i>	<i>sex</i>	6.77
<i>stock</i>	<i>jaguar</i>	0.92
<i>money</i>	<i>cash</i>	9.15
<i>development</i>	<i>issue</i>	3.97
<i>lad</i>	<i>brother</i>	4.46

Table 14.4: Subset of the WS-353 (Finkelstein et al., 2002) dataset of word similarity ratings (examples from Faruqui et al. (2016)).

7366 For any embedding method, we can evaluate whether the cosine similarity of word em-  
 7367 beddings is correlated with human judgments of word similarity. The WS-353 dataset (Finkel-  
 7368 stein et al., 2002) includes similarity scores for 353 word pairs (Table 14.4). To test the  
 7369 accuracy of embeddings for rare and morphologically complex words, Luong et al. (2013)  
 7370 introduce a dataset of “rare words.” Outside of English, word similarity resources are  
 7371 limited, mainly consisting of translations of WS-353.

7372 Word analogies (e.g., *king:queen :: man:woman*) have also been used to evaluate word  
 7373 embeddings (Mikolov et al., 2013). In this evaluation, the system is provided with the first  
 7374 three parts of the analogy ( $i_1 : j_1 :: i_2 : ?$ ), and the final element is predicted by finding the  
 7375 word embedding most similar to  $\mathbf{u}_{i_1} - \mathbf{u}_{j_1} + \mathbf{u}_{i_2}$ . Another evaluation tests whether word  
 7376 embeddings are related to broad lexical semantic categories called **supersenses** (Caramita  
 7377 and Johnson, 2003): verbs of motion, nouns that describe animals, nouns that describe  
 7378 body parts, and so on. These supersenses are annotated for English synsets in Word-  
 7379 Net (Fellbaum, 2010). This evaluation is implemented in the `qvec` metric, which tests  
 7380 whether the matrix of supersenses can be reconstructed from the matrix of word embed-  
 7381 dings (Tsvetkov et al., 2015).

7382 Levy et al. (2015) compared several dense word representations for English — includ-  
 7383 ing latent semantic analysis, WORD2VEC, and GloVe — using six word similarity metrics  
 7384 and two analogy tasks. None of the embeddings outperformed the others on every task,  
 7385 but skipgrams were the most broadly competitive. Hyperparameter tuning played a key  
 7386 role: any method will perform badly if the wrong hyperparameters are used. Relevant  
 7387 hyperparameters include the embedding size, as well as algorithm-specific details such  
 7388 as the neighborhood size and the number of negative samples.

### 7389 14.6.2 Extrinsic evaluations

7390 Word representations contribute to downstream tasks like sequence labeling and docu-  
 7391 ment classification by enabling generalization across words. The use of distributed repre-  
 7392 sentations as features is a form of **semi-supervised learning**, in which performance on a

7393 supervised learning problem is augmented by learning distributed representations from  
 7394 unlabeled data (Miller et al., 2004; Koo et al., 2008; Turian et al., 2010). These **pre-trained**  
 7395 **word representations** can be used as features in a linear prediction model, or as the input  
 7396 layer in a neural network, such as a Bi-LSTM tagging model (§ 7.6). Word representations  
 7397 can be evaluated by the performance of the downstream systems that consume them:  
 7398 for example, GloVe embeddings are convincingly better than Latent Semantic Analysis  
 7399 as features in the downstream task of named entity recognition (Pennington et al., 2014).  
 7400 Unfortunately, extrinsic and intrinsic evaluations do not always point in the same direc-  
 7401 tion, and the best word representations for one downstream task may perform poorly on  
 7402 another task (Schnabel et al., 2015).

7403 When word representations are updated from labeled data in the downstream task,  
 7404 they are said to be **fine-tuned**. When labeled data is plentiful, pre-training may be un-  
 7405 necessary; when labeled data is scarce, fine-tuning may lead to overfitting. Various com-  
 7406 binations of pre-training and fine-tuning can be employed. Pre-trained embeddings can  
 7407 be used as initialization before fine-tuning, and this can substantially improve perfor-  
 7408 mance (Lample et al., 2016). Alternatively, both fine-tuned and pre-trained embeddings  
 7409 can be used as inputs in a single model (Kim, 2014).

7410 In semi-supervised scenarios, pretrained word embeddings can be replaced by “con-  
 7411 textualized” word representations (Peters et al., 2018). These contextualized represen-  
 7412 tations are set to the hidden states of a deep bi-directional LSTM, which is trained as a  
 7413 bi-directional language model, motivating the name **ELMo (embeddings from language**  
 7414 **models**). Given a supervised learning problem, the language model generates contextu-  
 7415 alized representations, which are then used as the base layer in a task-specific supervised  
 7416 neural network. This approach yields significant gains over pretrained word embeddings  
 7417 on several tasks, presumably because the contextualized embeddings use unlabeled data  
 7418 to learn how to integrate linguistic context into the base layer of the supervised neural  
 7419 network.

## 7420 14.7 Distributed representations beyond distributional statistics

7421 Distributional word representations can be estimated from huge unlabeled datasets, thereby  
 7422 covering many words that do not appear in labeled data: for example, GloVe embeddings  
 7423 are estimated from 800 billion tokens of web data,<sup>3</sup> while the largest labeled datasets for  
 7424 NLP tasks are on the order of millions of tokens. Nonetheless, even a dataset of hundreds  
 7425 of billions of tokens will not cover every word that may be encountered in the future.  
 7426 Furthermore, many words will appear only a few times, making their embeddings un-  
 7427 reliable. Many languages exceed English in morphological complexity, and thus have  
 7428 lower token-to-type ratios. When this problem is coupled with small training corpora, it

---

<sup>3</sup><http://commoncrawl.org/>

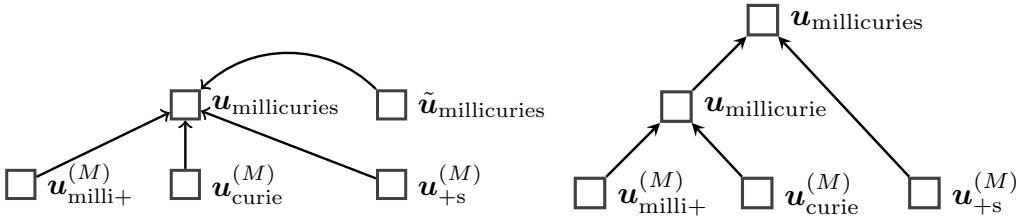


Figure 14.5: Two architectures for building word embeddings from subword units. On the left, morpheme embeddings  $u^{(m)}$  are combined by addition with the non-compositional word embedding  $\tilde{u}$  (Botha and Blunsom, 2014). On the right, morpheme embeddings are combined in a recursive neural network (Luong et al., 2013).

7429 becomes especially important to leverage other sources of information beyond distributional statistics.  
7430

#### 7431 14.7.1 Word-internal structure

7432 One solution is to incorporate word-internal structure into word embeddings. Purely  
7433 distributional approaches consider words as atomic units, but in fact, many words have  
7434 internal structure, so that their meaning can be **composed** from the representations of  
7435 sub-word units. Consider the following terms, all of which are missing from Google's  
7436 pre-trained WORD2VEC embeddings:<sup>4</sup>

7437 **millicuries** This word has **morphological** structure (see § 9.1.2 for more on morphology):  
7438 the prefix *milli-* indicates an amount, and the suffix *-s* indicates a plural. (A *millicurie*  
7439 is an unit of radioactivity.)

7440 **caesium** This word is a single morpheme, but the characters *-ium* are often associated  
7441 with chemical elements. (*Caesium* is the British spelling of a chemical element,  
7442 spelled *cesium* in American English.)

7443 **IAEA** This term is an acronym, as suggested by the use of capitalization. The prefix *I-* fre-  
7444 quently refers to international organizations, and the suffix *-A* often refers to agen-  
7445 cies or associations. (*IAEA* is the International Atomic Energy Agency.)

7446 **Zhezhgan** This term is in title case, suggesting the name of a person or place, and the  
7447 character bigram *zh* indicates that it is likely a transliteration. (*Zhezhgan* is a mining  
7448 facility in Kazakhstan.)

---

<sup>4</sup><https://code.google.com/archive/p/word2vec/>, accessed September 20, 2017

7449 How can word-internal structure be incorporated into word representations? One  
7450 approach is to construct word representations from embeddings of the characters or mor-  
7451 phemes. For example, if word  $i$  has morphological segments  $\mathcal{M}_i$ , then its embedding can  
7452 be constructed by addition (Botha and Blunsom, 2014),

$$\mathbf{u}_i = \tilde{\mathbf{u}}_i + \sum_{j \in \mathcal{M}_i} \mathbf{u}_j^{(M)}, \quad [14.27]$$

7453 where  $\mathbf{u}_m^{(M)}$  is a morpheme embedding and  $\tilde{\mathbf{u}}_i$  is a non-compositional embedding of the  
7454 whole word, which is an additional free parameter of the model (Figure 14.5, left side).  
7455 All embeddings are estimated from a **log-bilinear language model** (Mnih and Hinton,  
7456 2007), which is similar to the CBOW model (§ 14.5), but includes only contextual informa-  
7457 tion from preceding words. The morphological segments are obtained using an unsuper-  
7458 vised segmenter (Creutz and Lagus, 2007). For words that do not appear in the training  
7459 data, the embedding can be constructed directly from the morphemes, assuming that each  
7460 morpheme appears in some other word in the training data. The free parameter  $\tilde{\mathbf{u}}$  adds  
7461 flexibility: words with similar morphemes are encouraged to have similar embeddings,  
7462 but this parameter makes it possible for them to be different.

7463 Word-internal structure can be incorporated into word representations in various other  
7464 ways. Here are some of the main parameters.

7465 **Subword units.** Examples like *IAEA* and *Zhezhgan* are not based on morphological com-  
7466 position, and a morphological segmenter is unlikely to identify meaningful sub-  
7467 word units for these terms. Rather than using morphemes for subword embeddings,  
7468 one can use characters (Santos and Zadrozny, 2014; Ling et al., 2015; Kim et al., 2016),  
7469 character  $n$ -grams (Wieting et al., 2016; Bojanowski et al., 2017), and **byte-pair en-**  
7470 **codings**, a compression technique which captures frequent substrings (Gage, 1994;  
7471 Sennrich et al., 2016).

7472 **Composition.** Combining the subword embeddings by addition does not differentiate  
7473 between orderings, nor does it identify any particular morpheme as the **root**. A  
7474 range of more flexible compositional models have been considered, including re-  
7475 currence (Ling et al., 2015), convolution (Santos and Zadrozny, 2014; Kim et al.,  
7476 2016), and **recursive neural networks** (Luong et al., 2013), in which representa-  
7477 tions of progressively larger units are constructed over a morphological parse, e.g.  
7478  $((milli+curie)+s)$ ,  $((in+flam)+able)$ ,  $(in+(vis+ible))$ . A recursive embedding model is  
7479 shown in the right panel of Figure 14.5.

7480 **Estimation.** Estimating subword embeddings from a full dataset is computationally ex-  
7481 pensive. An alternative approach is to train a subword model to match pre-trained  
7482 word embeddings (Cotterell et al., 2016; Pinter et al., 2017). To train such a model, it  
7483 is only necessary to iterate over the vocabulary, and the not the corpus.

7484 **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,

$$\min_{\mathbf{U}} \sum_{j=1}^V \|\mathbf{u}_i - \hat{\mathbf{u}}_i\|_2 + \sum_{(i,j) \in \mathcal{L}} \beta_{ij} \|\mathbf{u}_i - \mathbf{u}_j\|_2, \quad [14.28]$$

7485 where  $\hat{\mathbf{u}}_i$  is the pretrained embedding of word  $i$ , and  $\mathcal{L} = \{(i,j)\}$  is a lexicon of word  
 7486 relations. The hyperparameter  $\beta_{ij}$  controls the importance of adjacent words having  
 7487 similar embeddings; Faruqui et al. (2015) set it to the inverse of the degree of word  $i$ ,  
 7488  $\beta_{ij} = |\{j : (i,j) \in \mathcal{L}\}|^{-1}$ . Retrofitting improves performance on a range of intrinsic evalua-  
 7489 tions, and gives small improvements on an extrinsic document classification task.

7490 **14.8 Distributed representations of multiword units**

7491 Can distributed representations extend to phrases, sentences, paragraphs, and beyond?  
 7492 Before exploring this possibility, recall the distinction between distributed and distri-  
 7493 butional representations. Neural embeddings such as WORD2VEC are both distributed  
 7494 (vector-based) and distributional (derived from counts of words in context). As we con-  
 7495 sider larger units of text, the counts decrease: in the limit, a multi-paragraph span of text  
 7496 would never appear twice, except by plagiarism. Thus, the meaning of a large span of  
 7497 text cannot be determined from distributional statistics alone; it must be computed com-  
 7498 positionally from smaller spans. But these considerations are orthogonal to the question  
 7499 of whether distributed representations — dense numerical vectors — are sufficiently ex-  
 7500 pressive to capture the meaning of phrases, sentences, and paragraphs.

7501 **14.8.1 Purely distributional methods**

7502 Some multiword phrases are non-compositional: the meaning of such phrases is not de-  
 7503 rived from the meaning of the individual words using typical compositional semantics.  
 7504 This includes proper nouns like *San Francisco* as well as idiomatic expressions like *kick*  
 7505 *the bucket* (Baldwin and Kim, 2010). For these cases, purely distributional approaches  
 7506 can work. A simple approach is to identify multiword units that appear together fre-  
 7507 quently, and then treat these units as words, learning embeddings using a technique such  
 7508 as WORD2VEC. The problem of identifying multiword units is sometimes called **colloca-**  
 7509 **tion extraction**, and can be approached using metrics such as pointwise mutual informa-  
 7510 tion: two-word units are extracted first, and then larger units are extracted. Mikolov et al.

7511 (2013) identify such units and then treat them as words when estimating skipgram em-  
7512 beddings, showing that the resulting embeddings perform reasonably on a task of solving  
7513 phrasal analogies, e.g. *New York : New York Times :: Baltimore : Baltimore Sun*.

### 7514 14.8.2 Distributional-compositional hybrids

7515 To move beyond short multiword phrases, composition is necessary. A simple but sur-  
7516 prisingly powerful approach is to represent a sentence with the average of its word em-  
7517 beddings (Mitchell and Lapata, 2010). This can be considered a hybrid of the distribu-  
7518 tional and compositional approaches to semantics: the word embeddings are computed  
7519 distributionally, and then the sentence representation is computed by composition.

7520 The WORD2VEC approach can be stretched considerably further, embedding entire  
7521 sentences using a model similar to skipgrams, in the “skip-thought” model of Kiros et al.  
7522 (2015). Each sentence is *encoded* into a vector using a recurrent neural network: the encod-  
7523 ing of sentence  $t$  is set to the RNN hidden state at its final token,  $h_{M_t}^{(t)}$ . This vector is then  
7524 a parameter in a *decoder* model that is used to generate the previous and subsequent sen-  
7525 tences: the decoder is another recurrent neural network, which takes the encoding of the  
7526 neighboring sentence as an additional parameter in its recurrent update. (This **encoder-**  
7527 **decoder model** is discussed at length in chapter 18.) The encoder and decoder are trained  
7528 simultaneously from a likelihood-based objective, and the trained encoder can be used to  
7529 compute a distributed representation of any sentence. Skip-thought can also be viewed  
7530 as a hybrid of distributional and compositional approaches: the vector representation of  
7531 each sentence is computed compositionally from the representations of the individual  
7532 words, but the training objective is distributional, based on sentence co-occurrence across  
7533 a corpus.

7534 **Autoencoders** are a variant of encoder-decoder models in which the decoder is trained  
7535 to produce the same text that was originally encoded, using only the distributed encod-  
7536 ing vector (Li et al., 2015). The encoding acts as a bottleneck, so that generalization is  
7537 necessary if the model is to successfully fit the training data. In **denoising autoencoders**,  
7538 the input is a corrupted version of the original sentence, and the auto-encoder must re-  
7539 construct the uncorrupted original (Vincent et al., 2010; Hill et al., 2016). By interpolating  
7540 between distributed representations of two sentences,  $\alpha \mathbf{u}_i + (1 - \alpha) \mathbf{u}_j$ , it is possible to gen-  
7541 erate sentences that combine aspects of the two inputs, as shown in Figure 14.6 (Bowman  
7542 et al., 2016).

7543 Autoencoders can also be applied to longer texts, such as paragraphs and documents.  
7544 This enables applications such as **question answering**, which can be performed by match-  
7545 ing the encoding of the question with encodings of candidate answers (Miao et al., 2016).

---

**this was the only way**  
 it was the only way  
 it was her turn to blink  
 it was hard to tell  
 it was time to move on  
 he had to do it again  
 they all looked at each other  
 they all turned to look back  
 they both turned to face him  
**they both turned and walked away**

---

Figure 14.6: By interpolating between the distributed representations of two sentences (in bold), it is possible to generate grammatical sentences that combine aspects of both (Bowman et al., 2016)

#### 7546 14.8.3 Supervised compositional methods

7547 Given a supervision signal, such as a label describing the sentiment or meaning of a sen-  
 7548 tence, a wide range of compositional methods can be applied to compute a distributed  
 7549 representation that then predicts the label. The simplest is to average the embeddings  
 7550 of each word in the sentence, and pass this average through a feedforward neural net-  
 7551 work (Iyyer et al., 2015). Convolutional and recurrent neural networks go further, with  
 7552 the ability to effectively capturing multiword phenomena such as negation (Kalchbrenner  
 7553 et al., 2014; Kim, 2014; Li et al., 2015; Tang et al., 2015). Another approach is to incorpo-  
 7554 rate the syntactic structure of the sentence into a **recursive neural networks**, in which the  
 7555 representation for each syntactic constituent is computed from the representations of its  
 7556 children (Socher et al., 2012). However, in many cases, recurrent neural networks perform  
 7557 as well or better than recursive networks (Li et al., 2015).

7558 Whether convolutional, recurrent, or recursive, a key question is whether supervised  
 7559 sentence representations are task-specific, or whether a single supervised sentence repre-  
 7560 sentation model can yield useful performance on other tasks. Wieting et al. (2015) train a  
 7561 variety of sentence embedding models for the task of labeling pairs of sentences as **para-**  
 7562 **phrases**. They show that the resulting sentence embeddings give good performance for  
 7563 sentiment analysis. The **Stanford Natural Language Inference corpus** classifies sentence  
 7564 pairs as **entailments** (the truth of sentence  $i$  implies the truth of sentence  $j$ ), **contradictions**  
 7565 (the truth of sentence  $i$  implies the falsity of sentence  $j$ ), and neutral ( $i$  neither entails nor  
 7566 contradicts  $j$ ). Sentence embeddings trained on this dataset transfer to a wide range of  
 7567 classification tasks (Conneau et al., 2017).

#### 7568 14.8.4 Hybrid distributed-symbolic representations

7569 The power of distributed representations is in their generality: the distributed representation  
7570 of a unit of text can serve as a summary of its meaning, and therefore as the input  
7571 for downstream tasks such as classification, matching, and retrieval. For example, dis-  
7572 tributed sentence representations can be used to recognize the paraphrase relationship  
7573 between closely related sentences like the following:

- 7574 (14.5) Donald thanked Vlad profusely.  
7575 (14.6) Donald conveyed to Vlad his profound appreciation.  
7576 (14.7) Vlad was showered with gratitude by Donald.

7577 Symbolic representations are relatively brittle to this sort of variation, but are better  
7578 suited to describe individual entities, the things that they do, and the things that are done  
7579 to them. In examples (14.5)-(14.7), we not only know that somebody thanked someone  
7580 else, but we can make a range of inferences about what has happened between the en-  
7581 tities named *Donald* and *Vlad*. Because distributed representations do not treat entities  
7582 symbolically, they lack the ability to reason about the roles played by entities across a sen-  
7583 tence or larger discourse.<sup>5</sup> A hybrid between distributed and symbolic representations  
7584 might give the best of both worlds: robustness to the many different ways of describing  
7585 the same event, plus the expressiveness to support inferences about entities and the roles  
7586 that they play.

7587 A “top-down” hybrid approach is to begin with logical semantics (of the sort de-  
7588 scribed in the previous two chapters), and but replace the predefined lexicon with a set  
7589 of distributional word clusters (Poon and Domingos, 2009; Lewis and Steedman, 2013). A  
7590 “bottom-up” approach is to add minimal symbolic structure to existing distributed repre-  
7591 sentations, such as vector representations for each entity (Ji and Eisenstein, 2015; Wiseman  
7592 et al., 2016). This has been shown to improve performance on two problems that we will  
7593 encounter in the following chapters: classification of **discourse relations** between adj-  
7594 cent sentences (chapter 16; Ji and Eisenstein, 2015), and **coreference resolution** of entity  
7595 mentions (chapter 15; Wiseman et al., 2016; Ji et al., 2017). Research on hybrid seman-  
7596 tic representations is still in an early stage, and future representations may deviate more  
7597 boldly from existing symbolic and distributional approaches.

#### 7598 Additional resources

7599 Turney and Pantel (2010) survey a number of facets of vector word representations, fo-  
7600 cusing on matrix factorization methods. Schnabel et al. (2015) highlight problems with

---

<sup>5</sup>At 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!”

7601 similarity-based evaluations of word embeddings, and present a novel evaluation that  
 7602 controls for word frequency. Baroni et al. (2014) address linguistic issues that arise in  
 7603 attempts to combine distributed and compositional representations.

7604 In bilingual and multilingual distributed representations, embeddings are estimated  
 7605 for translation pairs or tuples, such as (*dog, perro, chien*). These embeddings can improve  
 7606 machine translation (Zou et al., 2013; Klementiev et al., 2012), transfer natural language  
 7607 processing models across languages (Täckström et al., 2012), and make monolingual word  
 7608 embeddings more accurate (Faruqui and Dyer, 2014). A typical approach is to learn a pro-  
 7609 jection that maximizes the correlation of the distributed representations of each element  
 7610 in a translation pair, which can be obtained from a bilingual dictionary. Distributed rep-  
 7611 resentations can also be linked to perceptual information, such as image features. Bruni  
 7612 et al. (2014) use textual descriptions of images to obtain visual contextual information for  
 7613 various words, which supplements traditional distributional context. Image features can  
 7614 also be inserted as contextual information in log bilinear language models (Kiros et al.,  
 7615 2014), making it possible to automatically generate text descriptions of images.

## 7616 Exercises

- 7617 1. Prove that the sum of probabilities of paths through a hierarchical softmax tree is  
 7618 equal to one.
- 7619 2. In skipgram word embeddings, the negative sampling objective can be written as,

$$\mathcal{L} = \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{C}} \text{count}(i, j) \psi(i, j), \quad [14.29]$$

7619 with  $\psi(i, j)$  is defined in Equation 14.23.

7620 Suppose we draw the negative samples from the empirical unigram distribution  
 7621  $\hat{p}(i) = p_{\text{unigram}}(i)$ . First, compute the expectation of  $\mathcal{L}$  with respect to this probability.

7622 Next, take the derivative of this expectation with respect to the score of a single word  
 7623 context pair  $\sigma(\mathbf{u}_i \cdot \mathbf{v}_j)$ , and solve for the pointwise mutual information  $\text{PMI}(i, j)$ . You  
 7624 should be able to show that at the optimum, the PMI is a simple function of  $\sigma(\mathbf{u}_i \cdot \mathbf{v}_j)$   
 7625 and the number of negative samples.

- 7626 3. \* In Brown clustering, prove that the cluster merge that maximizes the average mu-  
 7627 tual information (Equation 14.13) also maximizes the log-likelihood objective (Equa-  
 7628 tion 14.12).
- 7629 4. A simple way to compute a distributed phrase representation is to add up the dis-  
 7630 tributed representations of the words in the phrase. Consider a sentiment analysis  
 7631 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:

$$\psi(\text{good}) > \psi(\text{not good}) \quad [14.30]$$

$$\psi(\text{bad}) < \psi(\text{not bad}). \quad [14.31]$$

7629 Then construct a similar example pair for the case in which phrase representations  
 7630 are the *average* of the word representations.

5. Now let's consider a slight modification to the prediction model in the previous problem:

$$\psi(\mathbf{w}) = \boldsymbol{\theta} \cdot \text{ReLU}\left(\sum_{m=1}^M \mathbf{x}_m\right) \quad [14.32]$$

7631 Show that in this case, it *is* possible to achieve the inequalities above. Your solution  
 7632 should provide the weights  $\boldsymbol{\theta}$  and the embeddings  $\mathbf{x}_{\text{good}}$ ,  $\mathbf{x}_{\text{bad}}$ , and  $\mathbf{x}_{\text{not}}$ .

7633 For the next two problems, download a set of pre-trained word embeddings, such as the  
 7634 WORD2VEC or polyglot embeddings.

- 7635 6. Use cosine similarity to find the most similar words to: *dog*, *whale*, *before*, *however*,  
 7636 *fabricate*.
- 7637 7. Use vector addition and subtraction to compute target vectors for the analogies below. After computing each target vector, find the top three candidates by cosine  
 7638 similarity.
- 7640 • *dog:puppy :: cat: ?*
  - 7641 • *speak:speaker :: sing: ?*
  - 7642 • *France:French :: England: ?*
  - 7643 • *France:wine :: England: ?*

7644 The remaining problems will require you to build a classifier and test its properties. Pick  
 7645 a multi-class text classification dataset, such as RCV1<sup>6</sup>). Divide your data into training  
 7646 (60%), development (20%), and test sets (20%), if no such division already exists.

---

<sup>6</sup>[http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/lyrl2004\\_rcv1v2\\_README.htm](http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm)

- 7647     8. Train a convolutional neural network, with inputs set to pre-trained word embed-  
7648         dings from the previous problem. Use a special, fine-tuned embedding for out-of-  
7649         vocabulary words. Train until performance on the development set does not im-  
7650         prove. You can also use the development set to tune the model architecture, such  
7651         as the convolution width and depth. Report *F-MEASURE* and accuracy, as well as  
7652         training time.
- 7653     9. Now modify your model from the previous problem to fine-tune the word embed-  
7654         dings. Report *F-MEASURE*, accuracy, and training time.
- 7655     10. Try a simpler approach, in which word embeddings in the document are averaged,  
7656         and then this average is passed through a feed-forward neural network. Again, use  
7657         the development data to tune the model architecture. How close is the accuracy to  
7658         the convolutional networks from the previous problems?

7659 

## Chapter 15

7660 

# Reference Resolution

7661 References are one of the most noticeable forms of linguistic ambiguity, afflicting not just  
7662 automated natural language processing systems, but also fluent human readers. Warnings  
7663 to avoid “ambiguous pronouns” are ubiquitous in manuals and tutorials on writing  
7664 style. But referential ambiguity is not limited to pronouns, as shown in the text in Fig-  
7665 ure 15.1. Each of the bracketed substrings refers to an entity that is introduced earlier  
7666 in the passage. These references include the pronouns *he* and *his*, but also the shortened  
7667 name *Cook*, and **nominals** such as *the firm* and *the firm’s biggest growth market*.

7668 **Reference resolution** subsumes several subtasks. This chapter will focus on **corefer-  
7669 ence resolution**, which is the task of grouping spans of text that refer to a single underly-  
7670 ing entity, or, in some cases, a single event: for example, the spans *Tim Cook*, *he*, and *Cook*  
7671 are all **coreferent**. These individual spans are called **mentions**, because they mention an  
7672 entity; the entity is sometimes called the **referent**. Each mention has a set of **antecedents**,  
7673 which are preceding mentions that are coreferent; for the first mention of an entity, the an-  
7674 tecedent set is empty. The task of **pronominal anaphora resolution** requires identifying  
7675 only the antecedents of pronouns. In **entity linking**, references are resolved not to other  
7676 spans of text, but to entities in a knowledge base. This task is discussed in chapter 17.

7677 Coreference resolution is a challenging problem for several reasons. Resolving differ-  
7678 ent types of **referring expressions** requires different types of reasoning: the features and  
7679 methods that are useful for resolving pronouns are different from those that are useful  
7680 to resolve names and nominals. Coreference resolution involves not only linguistic rea-  
7681 soning, but also world knowledge and pragmatics: you may not have known that China  
7682 was Apple’s biggest growth market, but it is likely that you effortlessly resolved this ref-  
7683 erence while reading the passage in Figure 15.1.<sup>1</sup> A further challenge is that coreference

---

<sup>1</sup>This 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.

7684 resolution decisions are often entangled: each mention adds information about the entity,  
 7685 which affects other coreference decisions. This means that coreference resolution must  
 7686 be addressed as a structure prediction problem. But as we will see, there is no dynamic  
 7687 program that allows the space of coreference decisions to be searched efficiently.

## 7688 15.1 Forms of referring expressions

7689 There are three main forms of referring expressions — pronouns, names, and nominals.

### 7690 15.1.1 Pronouns

7691 Pronouns are a closed class of words that are used for references. A natural way to think  
 7692 about pronoun resolution is SMASH (Kehler, 2007):

- 7693 • Search for candidate antecedents;  
 7694 • Match against hard agreement constraints;  
 7695 • And Select using Heuristics, which are “soft” constraints such as recency, syntactic  
 7696 prominence, and parallelism.

#### 7697 15.1.1.1 Search

7698 In the search step, candidate antecedents are identified from the preceding text or speech.<sup>2</sup>  
 7699 Any noun phrase can be a candidate antecedent, and pronoun resolution usually requires

2015).

<sup>2</sup>Pronouns 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.

7700 parsing the text to identify all such noun phrases.<sup>3</sup> Filtering heuristics can help to prune  
 7701 the search space to noun phrases that are likely to be coreferent (Lee et al., 2013; Durrett  
 7702 and Klein, 2013). In nested noun phrases, mentions are generally considered to be the  
 7703 largest unit with a given head word: thus, *Apple Inc. Chief Executive Tim Cook* would be  
 7704 included as a mention, but *Tim Cook* would not, since they share the same head word,  
 7705 *Cook*.

#### 7706 15.1.1.2 Matching constraints for pronouns

7707 References and their antecedents must agree on semantic features such as number, person,  
 7708 gender, and animacy. Consider the pronoun *he* in this passage from the running example:

7709 (15.2) Tim Cook has jetted in for talks with officials as [he] seeks to clear up a pile of  
 7710 problems...

7711 The pronoun and possible antecedents have the following features:

- 7712 • *he*: singular, masculine, animate, third person
- 7713 • *officials*: plural, animate, third person
- 7714 • *talks*: plural, inanimate, third person
- 7715 • *Tim Cook*: singular, masculine, animate, third person

7716 The SMASH method searches backwards from *he*, discarding *officials* and *talks* because they  
 7717 do not satisfy the agreements constraints.

7718 Another source of constraints comes from syntax — specifically, from the phrase struc-  
 7719 ture trees discussed in chapter 10. Consider a parse tree in which both *x* and *y* are phrasal  
 7720 constituents. The constituent *x* **c-commands** the constituent *y* iff the first branching node  
 7721 above *x* also dominates *y*. For example, in Figure 15.2a, *Abigail* c-commands *her*, because  
 7722 the first branching node above *Abigail*, *S*, also dominates *her*. Now, if *x* c-commands *y*,  
 7723 **government and binding theory** (Chomsky, 1982) states that *y* can refer to *x* only if it is  
 7724 a **reflexive pronoun** (e.g., *herself*). Furthermore, if *y* is a reflexive pronoun, then its an-  
 7725 tecedent must c-command it. Thus, in Figure 15.2a, *her* cannot refer to *Abigail*; conversely,  
 7726 if we replace *her* with *herself*, then the reflexive pronoun *must* refer to *Abigail*, since this is  
 7727 the only candidate antecedent that c-commands it.

7728 Now consider the example shown in Figure 15.2b. Here, *Abigail* does not c-command  
 7729 *her*, but *Abigail's mom* does. Thus, *her* can refer to *Abigail* — and we cannot use reflexive

---

<sup>3</sup>In 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.

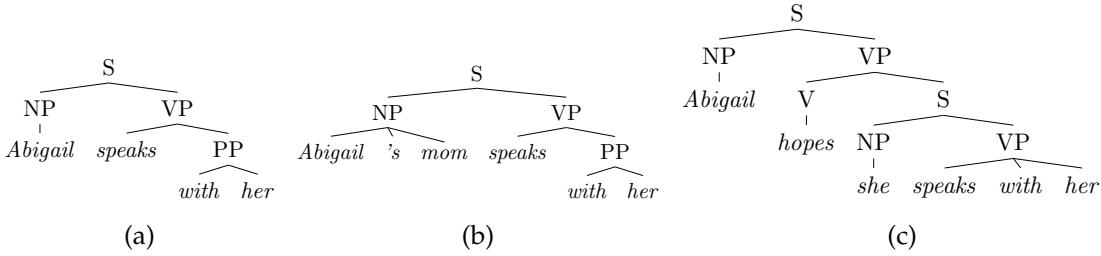


Figure 15.2: In (a), *Abigail* c-commands *her*; in (b), *Abigail* does not c-command *her*, but *Abigail's mom* does; in (c), the scope of *Abigail* is limited by the S non-terminal, so that *she* or *her* can bind to *Abigail*, but not both.

7730 *herself* in this context, unless we are talking about *Abigail*'s mom. However, *her* does not  
 7731 have to refer to *Abigail*. Finally, Figure 15.2c shows how these constraints are limited.  
 7732 In this case, the pronoun *she* can refer to *Abigail*, because the S non-terminal puts *Abigail*  
 7733 outside the domain of *she*. Similarly, *her* can also refer to *Abigail*. But *she* and *her* cannot be  
 7734 coreferent, because *she* c-commands *her*.

#### 7735 15.1.1.3 Heuristics

7736 After applying constraints, heuristics are applied to select among the remaining candidates.  
 7737 Recency is a particularly strong heuristic. All things equal, readers will prefer  
 7738 the more recent referent for a given pronoun, particularly when comparing referents that  
 7739 occur in different sentences. Jurafsky and Martin (2009) offer the following example:

- 7740 (15.3) The doctor found an old map in the captain's chest. Jim found an even older map  
 7741 hidden on the shelf. [It] described an island.

7742 Readers are expected to prefer the older map as the referent for the pronoun *it*.

7743 However, subjects are often preferred over objects, and this can contradict the preference  
 7744 for recency when two candidate referents are in the same sentence. For example,

- 7745 (15.4) Asha loaned Mei a book on Spanish. [She] is always trying to help people.

7746 Here, we may prefer to link *she* to *Asha* rather than *Mei*, because of *Asha*'s position in the  
 7747 subject role of the preceding sentence. (Arguably, this preference would not be strong  
 7748 enough to select *Asha* if the second sentence were *She is visiting Valencia next month*.)

7749 A third heuristic is parallelism:

- 7750 (15.5) Asha loaned Mei a book on Spanish. Olya loaned [her] a book on Portuguese.

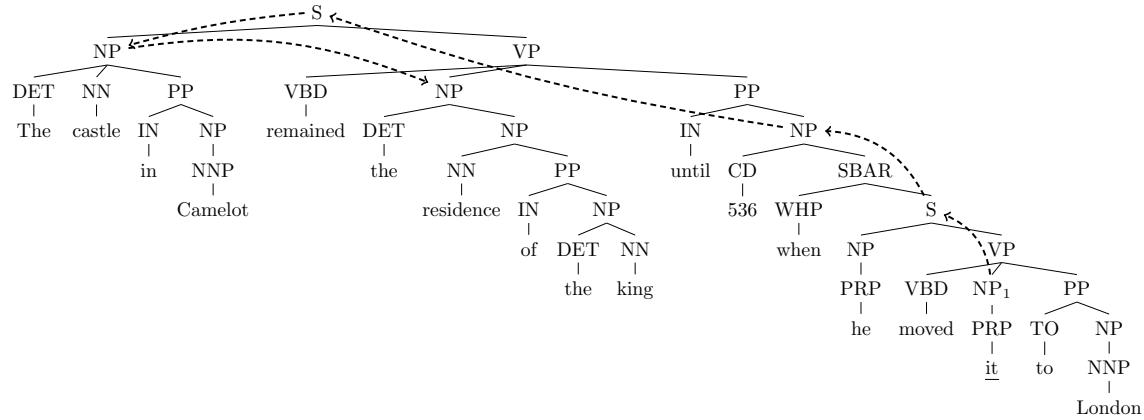


Figure 15.3: Left-to-right breadth-first tree traversal (Hobbs, 1978), indicating that the search for an antecedent for *it* ( $NP_1$ ) would proceed in the following order: 536; *the castle in Camelot*; *the residence of the king*; *Camelot*; *the king*. Hobbs (1978) proposes semantic constraints to eliminate 536 and *the castle in Camelot* as candidates, since they are unlikely to be the direct object of the verb *move*.

7751 Here *Mei* is preferred as the referent for *her*, contradicting the preference for the subject  
 7752 *Asha* in the preceding sentence.

7753 The recency and subject role heuristics can be unified by traversing the document in  
 7754 a syntax-driven fashion (Hobbs, 1978): each preceding sentence is traversed breadth-first,  
 7755 left-to-right (Figure 15.3). This heuristic successfully handles (15.4): *Asha* is preferred as  
 7756 the referent for *she* because the subject NP is visited first. It also handles (15.3): the older  
 7757 map is preferred as the referent for *it* because the more recent sentence is visited first. (An  
 7758 alternative unification of recency and syntax is proposed by **centering theory** (Grosz et al.,  
 7759 1995), which is discussed in detail in chapter 16.)

7760 In early work on reference resolution, the number of heuristics was small enough that  
 7761 a set of numerical weights could be set by hand (Lappin and Leass, 1994). More recent  
 7762 work uses machine learning to quantify the importance of each of these factors. However,  
 7763 pronoun resolution cannot be completely solved by constraints and heuristics alone. This  
 7764 is shown by the classic example pair (Winograd, 1972):

7765 (15.6) The [city council] denied [the protesters] a permit because [they] advocated / feared  
 7766 violence.

7767 Without reasoning about the motivations of the city council and protesters, it is unlikely  
 7768 that any system could correctly resolve both versions of this example.

7769 **15.1.1.4 Non-referential pronouns**

7770 While pronouns are generally used for reference, they need not refer to entities. The fol-  
 7771 lowing examples show how pronouns can refer to propositions, events, and speech acts.

- 7772 (15.7) They told me that I was too ugly for show business, but I didn't believe [it].  
 7773 (15.8) Asha saw Babak get angry, and I saw [it] too.  
 7774 (15.9) Asha said she worked in security. I suppose [that]'s one way to put it.

7775 These forms of reference are generally not annotated in large-scale coreference resolution  
 7776 datasets such as OntoNotes (Pradhan et al., 2011).

7777 Pronouns may also have **generic referents**:

- 7778 (15.10) A poor carpenter blames [her] tools.  
 7779 (15.11) On the moon, [you] have to carry [your] own oxygen.  
 7780 (15.12) Every farmer who owns a donkey beats [it]. (Geach, 1962)

7781 In the OntoNotes dataset, coreference is not annotated for generic referents, even in cases  
 7782 like these examples, in which the same generic entity is mentioned multiple times.

7783 Some pronouns do not refer to anything at all:

- 7784 (15.13) *[It]'s raining.*  
 [Il] pleut. (Fr)  
 7785 (15.14) [It] 's money that she's really after.  
 7786 (15.15) [It] is too bad that we have to work so hard.

7787 How can we automatically distinguish these usages of *it* from referential pronouns?  
 7788 Consider the the difference between the following two examples (Bergsma et al., 2008):

- 7789 (15.16) You can make [it] in advance.  
 7790 (15.17) You can make [it] in showbiz.

7791 In the second example, the pronoun *it* is non-referential. One way to see this is by substi-  
 7792 tuting another pronoun, like *them*, into these examples:

- 7793 (15.18) You can make [them] in advance.  
 7794 (15.19) ? You can make [them] in showbiz.

7795 The questionable grammaticality of the second example suggests that *it* is not referential.  
 7796 Bergsma et al. (2008) operationalize this idea by comparing distributional statistics for the

7797 *n*-grams around the word *it*, testing how often other pronouns or nouns appear in the  
7798 same context. In cases where nouns and other pronouns are infrequent, the *it* is unlikely  
7799 to be referential.

7800 **15.1.2 Proper Nouns**

7801 If a proper noun is used as a referring expression, it often corefers with another proper  
7802 noun, so that the coreference problem is simply to determine whether the two names  
7803 match. Subsequent proper noun references often use a shortened form, as in the running  
7804 example (Figure 15.1):

7805 (15.20) Apple Inc Chief Executive [Tim Cook] has jetted into China ... [Cook] is on his  
7806 first business trip to the country ...

7807 A typical solution for proper noun coreference is to match the syntactic **head words**  
7808 of the reference with the referent. In § 10.5.2, we saw that the head word of a phrase can  
7809 be identified by applying head percolation rules to the phrasal parse tree; alternatively,  
7810 the head can be identified as the root of the dependency subtree covering the name. For  
7811 sequences of proper nouns, the head word will be the final token.

7812 There are a number of caveats to the practice of matching head words of proper nouns.

- 7813 • In the European tradition, family names tend to be more specific than given names,  
7814 and family names usually come last. However, other traditions have other practices:  
7815 for example, in Chinese names, the family name typically comes first; in Japanese,  
7816 honorifics come after the name, as in *Nobu-San* (*Mr. Nobu*).
- 7817 • In organization names, the head word is often not the most informative, as in *Georgia*  
7818 *Tech* and *Virginia Tech*. Similarly, *Lebanon* does not refer to the same entity as *Southern Lebanon*, necessitating special rules for the specific case of geographical modi-  
7819 fiers (Lee et al., 2011).
- 7820 • Proper nouns can be nested, as in [*the CEO of [Microsoft]*], resulting in head word  
7821 match without coreference.

7823 Despite these difficulties, proper nouns are the easiest category of references to re-  
7824 solve (Stoyanov et al., 2009). In machine learning systems, one solution is to include a  
7825 range of matching features, including exact match, head match, and string inclusion. In  
7826 addition to matching features, competitive systems (e.g., Bengtson and Roth, 2008) in-  
7827 clude large lists, or **gazetteers**, of acronyms (e.g., *the National Basketball Association/NBA*),  
7828 demonyms (e.g., *the Israelis/Israel*), and other aliases (e.g., *the Georgia Institute of Technol-*  
7829 *ogy/Georgia Tech*).

7830 **15.1.3 Nominals**

7831 In coreference resolution, noun phrases that are neither pronouns nor proper nouns are  
 7832 referred to as **nominals**. In the running example (Figure 15.1), nominal references include:  
 7833 *the firm (Apple Inc); the firm's biggest growth market (China); and the country (China)*.

7834 Nominals are especially difficult to resolve (Denis and Baldridge, 2007; Durrett and  
 7835 Klein, 2013), and the examples above suggest why this may be the case: world knowledge  
 7836 is required to identify *Apple Inc* as a *firm*, and *China* as a *growth market*. Other difficult  
 7837 examples include the use of colloquial expressions, such as coreference between *Clinton*  
 7838 *campaign officials* and *the Clinton camp* (Soon et al., 2001).

7839 **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.<sup>4</sup> In the running example from Figure 15.1, the ground truth coreference annotation is:

$$c_1 = \{Apple\ Inc_{1:2}, the\ firm_{27:28}\} \quad [15.1]$$

$$c_2 = \{Apple\ Inc\ Chief\ Executive\ Tim\ Cook_{1:6}, he_{17}, Cook_{33}, his_{36}\} \quad [15.2]$$

$$c_3 = \{China_{10}, the\ firm\ 's\ biggest\ growth\ market_{27:32}, the\ country_{40:41}\} \quad [15.3]$$

7840 Each row specifies the token spans that mention an entity. (“Singleton” entities, which are  
 7841 mentioned only once (e.g., *talks, government officials*), are excluded from the annotations.)  
 7842 Equivalently, if given a set of  $M$  mentions,  $\{m_i\}_{i=1}^M$ , each mention  $i$  can be assigned to a  
 7843 cluster  $z_i$ , where  $z_i = z_j$  if  $i$  and  $j$  are coreferent. The cluster assignments  $z$  are invariant  
 7844 under permutation. The unique clustering associated with the assignment  $z$  is written  
 7845  $c(z)$ .

7846 **Mention identification** The task of identifying mention spans for coreference resolution  
 7847 is often performed by applying a set of heuristics to the phrase structure parse of each  
 7848 sentence. A typical approach is to start with all noun phrases and named entities, and  
 7849 then apply filtering rules to remove nested noun phrases with the same head (e.g., [*Apple*  
 7850 *CEO [Tim Cook]*]), numeric entities (e.g., [*100 miles*], [*97%*]), non-referential *it*, etc (Lee  
 7851 et al., 2013; Durrett and Klein, 2013). In general, these deterministic approaches err in  
 7852 favor of recall, since the mention clustering component can choose to ignore false positive  
 7853 mentions, but cannot recover from false negatives. An alternative is to consider all spans

---

<sup>4</sup>In 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.

7854 (up to some finite length) as candidate mentions, performing mention identification and  
 7855 clustering jointly (Daumé III and Marcu, 2005; Lee et al., 2017).

7856 **Mention clustering** The overwhelming majority of research on coreference resolution  
 7857 addresses the subtask of mention clustering, and this will be the focus of the remainder of  
 7858 this chapter. There are two main sets of approaches. In *mention-based models*, the scoring  
 7859 function for a coreference clustering decomposes over pairs of mentions. These pairwise  
 7860 decisions are then aggregated, using a clustering heuristic. Mention-based coreference  
 7861 clustering can be treated as a fairly direct application of supervised classification or rank-  
 7862 ing. However, the mention-pair locality assumption can result in incoherent clusters, like  
 7863  $\{Hillary Clinton \leftarrow Clinton \leftarrow Mr Clinton\}$ , in which the pairwise links score well, but the  
 7864 overall result is unsatisfactory. *Entity-based models* address this issue by scoring entities  
 7865 holistically. This can make inference more difficult, since the number of possible entity  
 7866 groupings is exponential in the number of mentions.

### 7867 15.2.1 Mention-pair models

7868 In the **mention-pair model**, a binary label  $y_{i,j} \in \{0, 1\}$  is assigned to each pair of mentions  
 7869 ( $i, j$ ), where  $i < j$ . If  $i$  and  $j$  corefer ( $z_i = z_j$ ), then  $y_{i,j} = 1$ ; otherwise,  $y_{i,j} = 0$ . The  
 7870 mention *he* in Figure 15.1 is preceded by five other mentions: (1) *Apple Inc*; (2) *Apple Inc*  
 7871 *Chief Executive Tim Cook*; (3) *China*; (4) *talks*; (5) *government officials*. The correct mention  
 7872 pair labeling is  $y_{2,6} = 1$  and  $y_{i \neq 2,6} = 0$  for all other  $i$ . If a mention  $j$  introduces a new entity,  
 7873 such as mention 3 in the example, then  $y_{i,j} = 0$  for all  $i$ . The same is true for “mentions”  
 7874 that do not refer to any entity, such as non-referential pronouns. If mention  $j$  refers to an  
 7875 entity that has been mentioned more than once, then  $y_{i,j} = 1$  for all  $i < j$  that mention the  
 7876 referent.

7877 By transforming coreference into a set of binary labeling problems, the mention-pair  
 7878 model makes it possible to apply an off-the-shelf binary classifier (Soon et al., 2001). This  
 7879 classifier is applied to each mention  $j$  independently, searching backwards from  $j$  until  
 7880 finding an antecedent  $i$  which corefers with  $j$  with high confidence. After identifying a  
 7881 single **antecedent**, the remaining mention pair labels can be computed by transitivity: if  
 7882  $y_{i,j} = 1$  and  $y_{j,k} = 1$ , then  $y_{i,k} = 1$ .

7883 Since the ground truth annotations give entity chains  $c$  but not individual mention-  
 7884 pair labels  $y$ , an additional heuristic must be employed to convert the labeled data into  
 7885 training examples for classification. A typical approach is to generate at most one pos-  
 7886 itive labeled instance  $y_{a_j,j} = 1$  for mention  $j$ , where  $a_j$  is the index of the most recent  
 7887 antecedent,  $a_j = \max\{i : i < j \wedge z_i = z_j\}$ . Negative labeled instances are generated for  
 7888 all for all  $i \in \{a_j + 1, \dots, j\}$ . In the running example, the most recent antecedent of the  
 7889 pronoun *he* is  $a_6 = 2$ , so the training data would be  $y_{2,6} = 1$  and  $y_{3,6} = y_{4,6} = y_{5,6} = 0$ .

7890 The variable  $y_{1,6}$  is not part of the training data, because the first mention appears before  
 7891 the true antecedent  $a_6 = 2$ .

7892 **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  $i$ ,

$$\hat{a}_i = \operatorname{argmax}_{a \in \{\epsilon, 1, 2, \dots, i-1\}} \psi_M(a, i), \quad [15.4]$$

7893 where  $\psi_M(a, i)$  is a score for the mention pair  $(a, i)$ . If  $a = \epsilon$ , then mention  $i$  does not refer  
 7894 to any previously-introduced entity — it is not **anaphoric**. Mention-ranking is similar to  
 7895 the mention-pair model, but all candidates are considered simultaneously, and at most  
 7896 a single antecedent is selected. The mention-ranking model explicitly accounts for the  
 7897 possibility that mention  $i$  is not anaphoric, through the score  $\psi_M(\epsilon, i)$ . The determination  
 7898 of anaphoricity can be made by a special classifier in a preprocessing step, so that non- $\epsilon$   
 7899 antecedents are identified only for spans that are determined to be anaphoric (Denis and  
 7900 Baldridge, 2008).

7901 As a learning problem, ranking can be trained using the same objectives as in dis-  
 7902 criminative classification. For each mention  $i$ , we can define a gold antecedent  $a_i^*$ , and an  
 7903 associated loss, such as the hinge loss,  $\ell_i = (1 - \psi_M(a_i^*, i) + \psi_M(\hat{a}, i))_+$  or the negative  
 7904 log-likelihood,  $\ell_i = -\log p(a_i^* | i; \theta)$ . (For more on learning to rank, see § 17.1.1.) But as  
 7905 with the mention-pair model, there is a mismatch between the labeled data, which comes  
 7906 in the form of mention sets, and the desired supervision, which would indicate the spe-  
 7907 cific antecedent of each mention. The antecedent variables  $\{a_i\}_{i=1}^M$  relate to the mention  
 7908 sets in a many-to-one mapping: each set of antecedents induces a single clustering, but a  
 7909 clustering can correspond to many different settings of antecedent variables.

A heuristic solution is to set  $a_i^* = \max\{j : j < i \wedge z_j = z_i\}$ , the most recent mention in  
 the same cluster as  $i$ . But the most recent mention may not be the most informative: in the  
 running example, the most recent antecedent of the mention *Cook* is the pronoun *he*, but  
 a more useful antecedent is the earlier mention *Apple Inc Chief Executive Tim Cook*. Rather  
 than selecting a specific antecedent to train on, the antecedent can be treated as a latent  
 variable, in the manner of the **latent variable perceptron** from § 12.4.2 (Fernandes et al.,

2014):

$$\hat{\mathbf{a}} = \operatorname{argmax}_{\mathbf{a}} \sum_{i=1}^M \psi_M(a_i, i) \quad [15.5]$$

$$\mathbf{a}^* = \operatorname{argmax}_{\mathbf{a} \in \mathcal{A}(c)} \sum_{i=1}^M \psi_M(a_i, i) \quad [15.6]$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \sum_{i=1}^M \frac{\partial L}{\partial \boldsymbol{\theta}} \psi_M(a_i^*, i) - \sum_{i=1}^M \frac{\partial L}{\partial \boldsymbol{\theta}} \psi_M(\hat{a}_i, i) \quad [15.7]$$

where  $\mathcal{A}(c)$  is the set of antecedent structures that is compatible with the ground truth coreference clustering  $c$ . Another alternative is to sum over all the conditional probabilities of antecedent structures that are compatible with the ground truth clustering (Durrett and Klein, 2013; Lee et al., 2017). For the set of mention  $\mathbf{m}$ , we compute the following probabilities:

$$p(c | \mathbf{m}) = \sum_{\mathbf{a} \in \mathcal{A}(c)} p(\mathbf{a} | \mathbf{m}) = \sum_{\mathbf{a} \in \mathcal{A}(c)} \prod_{i=1}^M p(a_i | i, \mathbf{m}) \quad [15.8]$$

$$p(a_i | i, \mathbf{m}) = \frac{\exp(\psi_M(a_i, i))}{\sum_{a' \in \{\epsilon, 1, 2, \dots, i-1\}} \exp(\psi_M(a', i))}. \quad [15.9]$$

7910 This objective rewards models that assign high scores for all valid antecedent structures.  
 7911 In the running example, this would correspond to summing the probabilities of the two  
 7912 valid antecedents for *Cook, he* and *Apple Inc Chief Executive Tim Cook*. In one of the exer-  
 7913 cises, you will compute the number of valid antecedent structures for a given clustering.

### 7914 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:

$$m_1 = \text{Hillary Clinton} \quad [15.10]$$

$$m_2 = \text{Clinton} \quad [15.11]$$

$$m_3 = \text{Bill Clinton} \quad [15.12]$$

7915 A mention-pair system might predict  $\hat{y}_{1,2} = 1, \hat{y}_{2,3} = 1, \hat{y}_{1,3} = 0$ . Similarly, a mention-  
 7916 ranking system might choose  $\hat{a}_2 = 1$  and  $\hat{a}_3 = 2$ . Logically, if mentions 1 and 3 are both  
 7917 coreferent with mention 2, then all three mentions must refer to the same entity. This  
 7918 constraint is known as **transitive closure**.

Transitive closure can be applied *post hoc*, revising the independent mention-pair or mention-ranking decisions. However, there are many possible ways to enforce transitive closure: in the example above, we could set  $\hat{y}_{1,3} = 1$ , or  $\hat{y}_{1,2} = 0$ , or  $\hat{y}_{2,3} = 0$ . For documents with many mentions, there may be many violations of transitive closure, and many possible fixes. Transitive closure can be enforced by always adding edges, so that  $\hat{y}_{1,3} = 1$  is preferred (e.g., Soon et al., 2001), but this can result in overclustering, with too many mentions grouped into too few entities.

Mention-pair coreference resolution can be viewed as a constrained optimization problem,

$$\begin{aligned} \max_{\mathbf{y} \in \{0,1\}^M} \quad & \sum_{j=1}^M \sum_{i=1}^j \psi_M(i, j) \times y_{i,j} \\ \text{s.t.} \quad & y_{i,j} + y_{j,k} - 1 \leq y_{i,k}, \quad \forall i < j < k, \end{aligned}$$

with the constraint enforcing transitive closure. This constrained optimization problem is equivalent to graph partitioning with positive and negative edge weights: construct a graph where the nodes are mentions, and the edges are the pairwise scores  $\psi_M(i, j)$ ; the goal is to partition the graph so as to maximize the sum of the edge weights between all nodes within the same partition (McCallum and Wellner, 2004). This problem is NP-hard, motivating approximations such as correlation clustering (Bansal et al., 2004) and **integer linear programming** (Klenner, 2007; Finkel and Manning, 2008, also see § 13.2.2).

#### 15.2.4 Entity-based models

A weakness of mention-based models is that they treat coreference resolution as a classification or ranking problem, when it is really a clustering problem: the goal is to group the mentions together into clusters that correspond to the underlying entities. Entity-based approaches attempt to identify these clusters directly. Such methods require a scoring function at the entity level, measuring whether each set of mentions is internally consistent. Coreference resolution can then be viewed as the following optimization,

$$\max_{\mathbf{z}} \quad \sum_{e=1} \psi_E(\{i : z_i = e\}), \tag{15.13}$$

where  $z_i$  indicates the entity referenced by mention  $i$ , and  $\psi_E(\{i : z_i = e\})$  is a scoring function applied to all mentions  $i$  that are assigned to entity  $e$ .

Entity-based coreference resolution is conceptually similar to the unsupervised clustering problems encountered in chapter 5: the goal is to obtain clusters of mentions that are internally coherent. The number of possible clusterings is the **Bell number**, which is

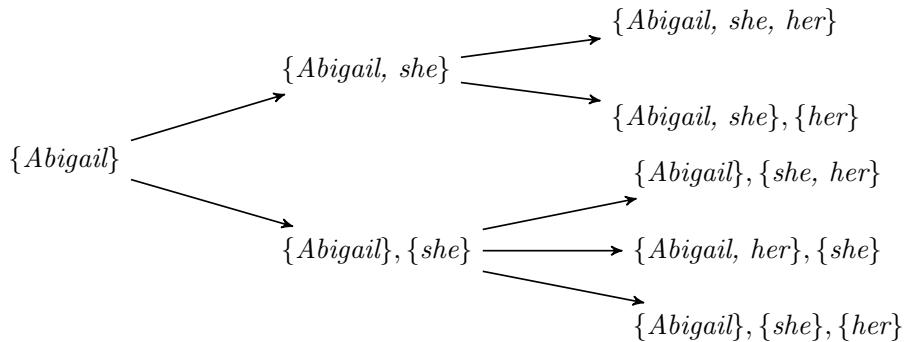


Figure 15.4: The Bell Tree for the sentence *Abigail hopes she speaks with her*. Which paths are excluded by the syntactic constraints mentioned in § 15.1.1?

defined by the following recurrence (Bell, 1934; Luo et al., 2004),

$$B_n = \sum_{k=0}^{n-1} B_k \binom{n-1}{k} = \frac{1}{e} \sum_{k=0}^{\infty} \frac{k^n}{k!}. \quad [15.14]$$

This recurrence is illustrated by the Bell tree, which is applied to a short coreference problem in Figure 15.4. The Bell number  $B_n$  grows exponentially with  $n$ , making exhaustive search of the space of clusterings impossible. For this reason, entity-based coreference resolution typically involves incremental search, in which clustering decisions are based on local evidence, in the hope of approximately optimizing the full objective in Equation 15.13. This approach is sometimes called **cluster ranking**, in contrast to mention ranking.

\***Generative models of coreference** Entity-based cooreference can be approached through probabilistic **generative models**, in which the mentions in the document are conditioned on a set of latent entities (Haghghi and Klein, 2007, 2010). An advantage of these methods is that they can be learned from unlabeled data (Poon and Domingos, 2008, e.g.); a disadvantage is that probabilistic inference is required not just for learning, but also for prediction. Furthermore, generative models require independence assumptions that are difficult to apply in coreference resolution, where the diverse and heterogeneous features do not admit an easy decomposition into mutually independent subsets.

#### 15.2.4.1 Incremental cluster ranking

The SMASH method (§ 15.1.1) can be extended to entity-based coreference resolution by building up coreference clusters while moving through the document (Cardie and Wagstaff, 1999). At each mention, the algorithm iterates backwards through possible antecedent

7955 clusters; but unlike SMASH, a cluster is selected only if *all* members of its cluster are compatible  
 7956 with the current mention. As mentions are added to a cluster, so are their features  
 7957 (e.g., gender, number, animacy). In this way, incoherent chains like *{Hillary Clinton, Clinton, Bill Clinton}*  
 7958 can be avoided. However, an incorrect assignment early in the document — a **search error**  
 7959 — might lead to a cascade of errors later on.

7960 More sophisticated search strategies can help to ameliorate the risk of search errors.  
 7961 One approach is **beam search** (§ 11.3), in which a set of hypotheses is maintained through-  
 7962 out search. Each hypothesis represents a path through the Bell tree (Figure 15.4). Hy-  
 7963 potheses are “expanded” either by adding the next mention to an existing cluster, or by  
 7964 starting a new cluster. Each expansion receives a score, based on Equation 15.13, and the  
 7965 top  $K$  hypotheses are kept on the beam as the algorithm moves to the next step.

7966 Incremental cluster ranking can be made more accurate by performing multiple passes  
 7967 over the document, applying rules (or “sieves”) with increasing recall and decreasing  
 7968 precision at each pass (Lee et al., 2013). In the early passes, coreference links are pro-  
 7969 posed only between mentions that are highly likely to corefer (e.g., exact string match  
 7970 for full names and nominals). Information can then be shared among these mentions,  
 7971 so that when more permissive matching rules are applied later, agreement is preserved  
 7972 across the entire cluster. For example, in the case of *{Hillary Clinton, Clinton, she}*, the  
 7973 name-matching sieve would link *Clinton* and *Hillary Clinton*, and the pronoun-matching  
 7974 sieve would then link *she* to the combined cluster. A deterministic multi-pass system  
 7975 won nearly every track of the 2011 CoNLL shared task on coreference resolution (Prad-  
 7976 han et al., 2011). Given the dominance of machine learning in virtually all other areas  
 7977 of natural language processing — and more than fifteen years of prior work on machine  
 7978 learning for coreference — this was a surprising result, even if learning-based methods  
 7979 have subsequently regained the upper hand (e.g., Lee et al., 2017, the state-of-the-art at  
 7980 the time of this writing).

#### 7981 15.2.4.2 Incremental perceptron

Incremental coreference resolution can be learned with the **incremental perceptron**, as described in § 11.3.2. At mention  $i$ , each hypothesis on the beam corresponds to a clustering of mentions  $1 \dots i - 1$ , or equivalently, a path through the Bell tree up to position  $i - 1$ . As soon as none of the hypotheses on the beam are compatible with the gold coreference clustering, a perceptron update is made (Daumé III and Marcu, 2005). For concreteness, consider a linear cluster ranking model,

$$\psi_E(\{i : z_i = e\}) = \sum_{i:z_i=e} \theta \cdot f(i, \{j : j < i \wedge z_j = e\}), \quad [15.15]$$

7982 where the score for each cluster is computed as the sum of scores of all mentions that are  
 7983 linked into the cluster, and  $f(i, \emptyset)$  is a set of features for the non-anaphoric mention that  
 7984 initiates the cluster.

7985 Using Figure 15.4 as an example, suppose that the ground truth is,

$$\mathbf{c}^* = \{\text{Abigail}, \text{her}\}, \{\text{she}\}, \quad [15.16]$$

7986 but that with a beam of size one, the learner reaches the hypothesis,

$$\hat{\mathbf{c}} = \{\text{Abigail}, \text{she}\}. \quad [15.17]$$

This hypothesis is incompatible with  $\mathbf{c}^*$ , so an update is needed:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{f}(\mathbf{c}^*) - \mathbf{f}(\hat{\mathbf{c}}) \quad [15.18]$$

$$= \boldsymbol{\theta} + (\mathbf{f}(\text{Abigail}, \emptyset) + \mathbf{f}(\text{she}, \emptyset)) - (\mathbf{f}(\text{Abigail}, \emptyset) + \mathbf{f}(\text{she}, \{\text{Abigail}\})) \quad [15.19]$$

$$= \boldsymbol{\theta} + \mathbf{f}(\text{she}, \emptyset) - \mathbf{f}(\text{she}, \{\text{Abigail}\}). \quad [15.20]$$

7987 This style of incremental update can also be applied to a margin loss between the gold  
 7988 clustering and the top clustering on the beam. By backpropagating from this loss, it is also  
 7989 possible to train a more complicated scoring function, such as a neural network in which  
 7990 the score for each entity is a function of embeddings for the entity mentions (Wiseman  
 7991 et al., 2015).

#### 7992 15.2.4.3 Reinforcement learning

7993 **Reinforcement learning** is a topic worthy of a textbook of its own (Sutton and Barto,  
 7994 1998),<sup>5</sup> so this section will provide only a very brief overview, in the context of coreference  
 7995 resolution. A stochastic **policy** assigns a probability to each possible **action**, conditional  
 7996 on the context. The goal is to learn a policy that achieves a high expected reward, or  
 7997 equivalently, a low expected cost.

7998 In incremental cluster ranking, a complete clustering on  $M$  mentions can be produced  
 7999 by a sequence of  $M$  actions, in which the action  $z_i$  either merges mention  $i$  with an existing  
 8000 cluster or begins a new cluster. We can therefore create a stochastic policy using the cluster  
 8001 scores (Clark and Manning, 2016),

$$\Pr(z_i = e; \boldsymbol{\theta}) = \frac{\exp \psi_E(i \cup \{j : z_j = e\}; \boldsymbol{\theta})}{\sum_{e'} \exp \psi_E(i \cup \{j : z_j = e'\}; \boldsymbol{\theta})}, \quad [15.21]$$

8002 where  $\psi_E(i \cup \{j : z_j = e\}; \boldsymbol{\theta})$  is the score under parameters  $\boldsymbol{\theta}$  for assigning mention  $i$  to  
 8003 cluster  $e$ . This score can be an arbitrary function of the mention  $i$ , the cluster  $e$  and its  
 8004 (possibly empty) set of mentions; it can also include the history of actions taken thus far.

---

<sup>5</sup>A 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).

8005 If a policy assigns probability  $p(c; \theta)$  to clustering  $c$ , then its expected loss is,

$$L(\theta) = \sum_{c \in \mathcal{C}(m)} p_\theta(c) \times \ell(c), \quad [15.22]$$

8006 where  $\mathcal{C}(m)$  is the set of possible clusterings for mentions  $m$ . The loss  $\ell(c)$  can be based on  
 8007 any arbitrary scoring function, including the complex evaluation metrics used in corefer-  
 8008 ence resolution (see § 15.4). This is an advantage of reinforcement learning, which can be  
 8009 trained directly on the evaluation metric — unlike traditional supervised learning, which  
 8010 requires a loss function that is differentiable and decomposable across individual deci-  
 8011 sions.

Rather than summing over the exponentially many possible clusterings, we can approximate the expectation by sampling trajectories of actions,  $z = (z_1, z_2, \dots, z_M)$ , from the current policy. Each action  $z_i$  corresponds to a step in the Bell tree: adding mention  $m_i$  to an existing cluster, or forming a new cluster. Each trajectory  $z$  corresponds to a single clustering  $c$ , and so we can write the loss of an action sequence as  $\ell(c(z))$ . The **policy gradient** algorithm computes the gradient of the expected loss as an expectation over trajectories (Sutton et al., 2000),

$$\frac{\partial}{\partial \theta} L(\theta) = E_{z \sim \mathcal{Z}(m)} \ell(c(z)) \sum_{i=1}^M \frac{\partial}{\partial \theta} \log p(z_i | z_{1:i-1}, m) \quad [15.23]$$

$$\approx \frac{1}{K} \sum_{k=1}^K \ell(c(z^{(k)})) \sum_{i=1}^M \frac{\partial}{\partial \theta} \log p(z_i^{(k)} | z_{1:i-1}^{(k)}, m) \quad [15.24]$$

[15.25]

8012 where the action sequence  $z^{(k)}$  is sampled from the current policy. Unlike the incremental  
 8013 perceptron, an update is not made until the complete action sequence is available.

#### 8014 15.2.4.4 Learning to search

8015 Policy gradient can suffer from high variance: while the average loss over  $K$  samples is  
 8016 asymptotically equal to the expected reward of a given policy, this estimate may not be  
 8017 accurate unless  $K$  is very large. This can make it difficult to allocate credit and blame to  
 8018 individual actions. In **learning to search**, this problem is addressed through the addition  
 8019 of an **oracle** policy, which is known to receive zero or small loss. The oracle policy can be  
 8020 used in two ways:

- 8021 • The oracle can be used to generate partial hypotheses that are likely to score well,  
 8022 by generating  $i$  actions from the initial state. These partial hypotheses are then used  
 8023 as starting points for the learned policy. This is known as **roll-in**.

**Algorithm 18** Learning to search for entity-based coreference resolution

---

```

1: procedure COMPUTE-GRADIENT(mentions  $m$ , loss function  $\ell$ , parameters  $\theta$ )
2:    $L(\theta) \leftarrow 0$ 
3:    $z \sim p(z | m; \theta)$                                  $\triangleright$  Sample a trajectory from the current policy
4:   for  $i \in \{1, 2, \dots, M\}$  do
5:     for action  $z \in \mathcal{Z}(z_{1:i-1}, m)$  do           $\triangleright$  All possible actions after history  $z_{1:i-1}$ 
6:        $h \leftarrow z_{1:i-1} \oplus z$                        $\triangleright$  Concatenate history  $z_{1:i-1}$  with action  $z$ 
7:       for  $j \in \{i+1, i+2, \dots, M\}$  do            $\triangleright$  Roll-out
8:          $h_j \leftarrow \operatorname{argmin}_h \ell(h_{1:j-1} \oplus h)$      $\triangleright$  Oracle selects action with minimum loss
9:        $L(\theta) \leftarrow L(\theta) + p(z | z_{1:i-1}, m; \theta) \times \ell(h)$        $\triangleright$  Update expected loss
10:      return  $\frac{\partial}{\partial \theta} L(\theta)$ 

```

---

- 8024 • The oracle can be used to compute the minimum possible loss from a given state, by  
 8025 generating  $M - i$  actions from the current state until completion. This is known as  
 8026 **roll-out**.

8027 The oracle can be combined with the existing policy during both roll-in and roll-out, sam-  
 8028 pling actions from each policy (Daumé III et al., 2009). One approach is to gradually  
 8029 decrease the number of actions drawn from the oracle over the course of learning (Ross  
 8030 et al., 2011).

8031 In the context of entity-based coreference resolution, Clark and Manning (2016) use  
 8032 the learned policy for roll-in and the oracle policy for roll-out. Algorithm 18 shows how  
 8033 the gradients on the policy weights are computed in this case. In this application, the  
 8034 oracle is “noisy”, because it selects the action that minimizes only the *local* loss — the  
 8035 accuracy of the coreference clustering up to mention  $i$  — rather than identifying the action  
 8036 sequence that will lead to the best final coreference clustering on the entire document.  
 8037 When learning from noisy oracles, it can be helpful to mix in actions from the current  
 8038 policy with the oracle during roll-out (Chang et al., 2015).

8039 **15.3 Representations for coreference resolution**

8040 Historically, coreference resolution has employed an array of hand-engineered features  
 8041 to capture the linguistic constraints and preferences described in § 15.1 (Soon et al., 2001).  
 8042 Later work has documented the utility of lexical and bilexical features on mention pairs (Björkelund  
 8043 and Nugues, 2011; Durrett and Klein, 2013). The most recent and successful methods re-  
 8044 place many (but not all) of these features with distributed representations of mentions  
 8045 and entities (Wiseman et al., 2015; Clark and Manning, 2016; Lee et al., 2017).

8046 **15.3.1 Features**

8047 Coreference features generally rely on a preprocessing pipeline to provide part-of-speech  
 8048 tags and phrase structure parses. This pipeline makes it possible to design features that  
 8049 capture many of the phenomena from § 15.1, and is also necessary for typical approaches  
 8050 to mention identification. However, the pipeline may introduce errors that propagate  
 8051 to the downstream coreference clustering system. Furthermore, the existence of such  
 8052 a pipeline presupposes resources such as treebanks, which do not exist for many lan-  
 8053 guages.<sup>6</sup>

8054 **15.3.1.1 Mention features**

8055 Features of individual mentions can help to predict anaphoricity. In systems where men-  
 8056 tion detection is performed jointly with coreference resolution, these features can also  
 8057 predict whether a span of text is likely to be a mention. For mention  $i$ , typical features  
 8058 include:

8059 **Mention type.** Each span can be identified as a pronoun, name, or nominal, using the  
 8060 part-of-speech of the head word of the mention: both the Penn Treebank and Uni-  
 8061 versal Dependencies tagsets (§ 8.1.1) include tags for pronouns and proper nouns,  
 8062 and all other heads can be marked as nominals (Haghghi and Klein, 2009).

8063 **Mention width.** The number of tokens in a mention is a rough predictor of its anaphor-  
 8064 icity, with longer mentions being less likely to refer back to previously-defined enti-  
 8065 ties.

8066 **Lexical features.** The first, last, and head words can help to predict anaphoricity; they are  
 8067 also useful in conjunction with features such as mention type and part-of-speech,  
 8068 providing a rough measure of agreement (Björkelund and Nugues, 2011). The num-  
 8069 ber of lexical features can be very large, so it can be helpful to select only frequently-  
 8070 occurring features (Durrett and Klein, 2013).

8071 **Morphosyntactic features.** These features include the part-of-speech, number, gender,  
 8072 and dependency ancestors.

8073 The features for mention  $i$  and candidate antecedent  $a$  can be conjoined, producing  
 8074 joint features that can help to assess the compatibility of the two mentions. For example,  
 8075 Durrett and Klein (2013) conjoin each feature with the mention types of the anaphora  
 8076 and the antecedent. Coreference resolution corpora such as ACE and OntoNotes contain

---

<sup>6</sup>The 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>

8077 documents from various genres. By conjoining the genre with other features, it is possible  
8078 to learn genre-specific feature weights.

8079 **15.3.1.2 Mention-pair features**

8080 For any pair of mentions  $i$  and  $j$ , typical features include:

8081 **Distance.** The number of intervening tokens, mentions, and sentences between  $i$  and  $j$   
8082 can all be used as distance features. These distances can be computed on the surface  
8083 text, or on a transformed representation reflecting the breadth-first tree traversal  
8084 (Figure 15.3). Rather than using the distances directly, they are typically binned,  
8085 creating binary features.

8086 **String match.** A variety of string match features can be employed: exact match, suffix  
8087 match, head match, and more complex matching rules that disregard irrelevant  
8088 modifiers (Soon et al., 2001).

8089 **Compatibility.** Building on the model, features can measure the anaphor and antecedent  
8090 agree with respect to morphosyntactic attributes such as gender, number, and ani-  
8091 macy.

8092 **Nesting.** If one mention is nested inside another (e.g., *[The President of [France]]*), they  
8093 generally cannot corefer.

8094 **Same speaker.** For documents with quotations, such as news articles, personal pronouns  
8095 can be resolved only by determining the speaker for each mention (Lee et al., 2013).  
8096 Coreference is also more likely between mentions from the same speaker.

8097 **Gazetteers.** These features indicate that the anaphor and candidate antecedent appear in  
8098 a gazetteer of acronyms (e.g., *USA/United States*, *GATech/Georgia Tech*), demonymns  
8099 (e.g., *Israel/Israeli*), or other aliases (e.g., *Knickerbockers/New York Knicks*).

8100 **Lexical semantics.** These features use a lexical resource such as WordNet to determine  
8101 whether the head words of the mentions are related through synonymy, antonymy,  
8102 and hypernymy (§ 4.2).

8103 **Dependency paths.** The dependency path between the anaphor and candidate antecedent  
8104 can help to determine whether the pair can corefer, under the government and bind-  
8105 ing constraints described in § 15.1.1.

8106 Comprehensive lists of mention-pair features are offered by Bengtson and Roth (2008) and  
8107 Rahman and Ng (2011). Neural network approaches use far fewer mention-pair features:  
8108 for example, Lee et al. (2017) include only speaker, genre, distance, and mention width  
8109 features.

8110 **Semantics** In many cases, coreference seems to require knowledge and semantic in-  
 8111 ferences, as in the running example, where we link *China* with a *country* and a *growth*  
 8112 *market*. Some of this information can be gleaned from WordNet, which defines a graph  
 8113 over **synsets** (see § 4.2). For example, one of the synsets of *China* is an instance of an  
 8114 *Asian\_nation#1*, which in turn is a hyponym of *country#2*, a synset that includes  
 8115 *country*.<sup>7</sup> Such paths can be used to measure the similarity between concepts (Pedersen  
 8116 et al., 2004), and this similarity can be incorporated into coreference resolution as a fea-  
 8117 ture (Ponzetto and Strube, 2006). Similar ideas can be applied to knowledge graphs in-  
 8118 duced from Wikipedia (Ponzetto and Strube, 2007). But while such approaches improve  
 8119 relatively simple classification-based systems, they have proven less useful when added  
 8120 to the current generation of techniques.<sup>8</sup> For example, Durrett and Klein (2013) employ  
 8121 a range of semantics-based features — WordNet synonymy and hypernymy relations on  
 8122 head words, named entity types (e.g., person, organization), and unsupervised clustering  
 8123 over nominal heads — but find that these features give minimal improvement over a  
 8124 baseline system using surface features.

### 8125 15.3.1.3 Entity features

8126 Many of the features for entity-mention coreference are generated by aggregating mention-  
 8127 pair features over all mentions in the candidate entity (Culotta et al., 2007; Rahman and  
 8128 Ng, 2011). Specifically, for each binary mention-pair feature  $f(i, j)$ , we compute the fol-  
 8129 lowing entity-mention features for mention  $i$  and entity  $e = \{j : j < i \wedge z_j = e\}$ .

- 8130 • ALL-TRUE: Feature  $f(i, j)$  holds for all mentions  $j \in e$ .
- 8131 • MOST-TRUE: Feature  $f(i, j)$  holds for at least half and fewer than all mentions  $j \in e$ .
- 8132 • MOST-FALSE: Feature  $f(i, j)$  holds for at least one and fewer than half of all men-  
 8133 tions  $j \in e$ .
- 8134 • NONE: Feature  $f(i, j)$  does not hold for any mention  $j \in e$ .

8135 For scalar mention-pair features (e.g., distance features), aggregation can be performed by  
 8136 computing the minimum, maximum, and median values across all mentions in the cluster.  
 8137 Additional entity-mention features include the number of mentions currently clustered in  
 8138 the entity, and ALL-X and MOST-X features for each mention type.

### 8139 15.3.2 Distributed representations of mentions and entities

8140 Recent work has emphasized distributed representations of both mentions and entities.  
 8141 One potential advantage is that pre-trained embeddings could help to capture the se-

---

<sup>7</sup>teletype font is used to indicate wordnet synsets, and *italics* is used to indicate strings.

<sup>8</sup>This 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.

8142 mantic compatibility underlying nominal coreference, helping with difficult cases like  
 8143 (*Apple, the firm*) and (*China, the firm's biggest growth market*). Furthermore, a distributed  
 8144 representation of entities can be trained to capture semantic features that are added by  
 8145 each mention.

8146 **15.3.2.1 Mention embeddings**

8147 Entity mentions can be embedded into a vector space, providing the base layer for neural  
 8148 networks that score coreference decisions (Wiseman et al., 2015).

8149 **Constructing the mention embedding** Various approaches for embedding multiword  
 8150 units can be applied (see § 14.8). Figure 15.5 shows a recurrent neural network approach,  
 8151 which begins by running a bidirectional LSTM over the entire text, obtaining hidden states  
 8152 from the left-to-right and right-to-left passes,  $\mathbf{h}_m = [\overleftarrow{\mathbf{h}}_m; \overrightarrow{\mathbf{h}}_m]$ . Each candidate mention  
 8153 span  $(s, t)$  is then represented by the vertical concatenation of four vectors:

$$\mathbf{u}^{(s,t)} = [\mathbf{u}_{\text{first}}^{(s,t)}; \mathbf{u}_{\text{last}}^{(s,t)}; \mathbf{u}_{\text{head}}^{(s,t)}; \phi^{(s,t)}], \quad [15.26]$$

8154 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  
 8155 embedding of the last word,  $\mathbf{u}_{\text{head}}^{(s,t)}$  is the embedding of the “head” word, and  $\phi^{(s,t)}$  is a  
 8156 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**:

$$\tilde{\alpha}_m = \theta_\alpha \cdot \mathbf{h}_m \quad [15.27]$$

$$\mathbf{a}^{(s,t)} = \text{SoftMax}([\tilde{\alpha}_{s+1}, \tilde{\alpha}_{s+2}, \dots, \tilde{\alpha}_t]) \quad [15.28]$$

$$\mathbf{u}_{\text{head}}^{(s,t)} = \sum_{m=s+1}^t a_m^{(s,t)} \mathbf{h}_m. \quad [15.29]$$

8157 Each token  $m$  gets a scalar score  $\tilde{\alpha}_m = \theta_\alpha \cdot \mathbf{h}_m$ , which is the dot product of the LSTM  
 8158 hidden state  $\mathbf{h}_m$  and a vector of weights  $\theta_\alpha$ . The vector of scores for tokens in the span  
 8159  $m \in \{s + 1, s + 2, \dots, t\}$  is then passed through a softmax layer, yielding a vector  $\mathbf{a}^{(s,t)}$   
 8160 that allocates one unit of attention across the span. This eliminates the need for syntactic  
 8161 parsing to recover the head word; instead, the model learns to identify the most important  
 8162 words in each span. Attention mechanisms were introduced in neural machine transla-  
 8163 tion (Bahdanau et al., 2014), and are described in more detail in § 18.3.1.

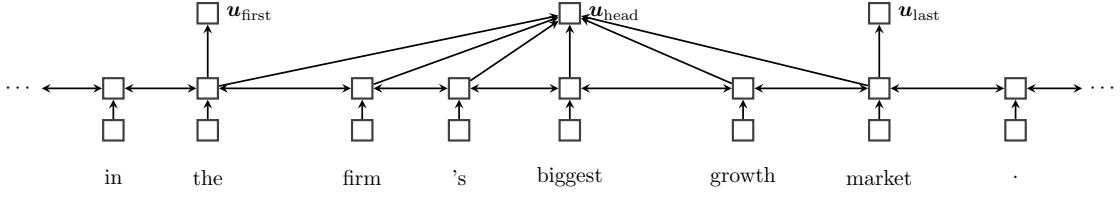


Figure 15.5: A bidirectional recurrent model of mention embeddings. The mention is represented by its first word, its last word, and an estimate of its head word, which is computed from a weighted average (Lee et al., 2017).

**Using mention embeddings** Given a set of mention embeddings, each mention  $i$  and candidate antecedent  $a$  is scored as,

$$\psi(a, i) = \psi_S(a) + \psi_S(i) + \psi_M(a, i) \quad [15.30]$$

$$\psi_S(a) = \text{FeedForward}_S(\mathbf{u}^{(a)}) \quad [15.31]$$

$$\psi_S(i) = \text{FeedForward}_S(\mathbf{u}^{(i)}) \quad [15.32]$$

$$\psi_M(a, i) = \text{FeedForward}_M([\mathbf{u}^{(a)}; \mathbf{u}^{(i)}; \mathbf{u}^{(a)} \odot \mathbf{u}^{(i)}; \mathbf{f}(a, i, \mathbf{w})]), \quad [15.33]$$

where  $\mathbf{u}^{(a)}$  and  $\mathbf{u}^{(i)}$  are the embeddings for spans  $a$  and  $i$  respectively, as defined in Equation 15.26.

- The scores  $\psi_S(a)$  quantify whether span  $a$  is likely to be a coreferring mention, independent of what it corefers with. This allows the model to learn identify mentions directly, rather than identifying mentions with a preprocessing step.
- The score  $\psi_M(a, i)$  computes the compatibility of spans  $a$  and  $i$ . Its base layer is a vector that includes the embeddings of spans  $a$  and  $i$ , their elementwise product  $\mathbf{u}^{(a)} \odot \mathbf{u}^{(i)}$ , and a vector of surface features  $\mathbf{f}(a, i, \mathbf{w})$ , including distance, speaker, and genre information.

Lee et al. (2017) provide an error analysis that shows how this method can correctly link a *blaze* and a *fire*, while incorrectly linking *pilots* and *fight attendants*. In each case, the coreference decision is based on similarities in the word embeddings.

Rather than embedding individual mentions, Clark and Manning (2016) embed mention pairs. At the base layer, their network takes embeddings of the words in and around each mention, as well as **one-hot** vectors representing a few surface features, such as the distance and string matching features. This base layer is then passed through a multilayer feedforward network with ReLU nonlinearities, resulting in a representation of the mention pair. The output of the mention pair encoder  $\mathbf{u}_{i,j}$  is used in the scoring function of a mention-ranking model,  $\psi_M(i, j) = \theta \cdot \mathbf{u}_{i,j}$ . A similar approach is used to score cluster

8183 pairs, constructing a cluster-pair encoding by **pooling** over the mention-pair encodings  
8184 for all pairs of mentions within the two clusters.

8185 **15.3.2.2 Entity embeddings**

8186 In entity-based coreference resolution, each entity should be represented by properties of  
8187 its mentions. In a distributed setting, we maintain a set of vector entity embeddings,  $v_e$ .  
8188 Each candidate mention receives an embedding  $u_i$ ; Wiseman et al. (2016) compute this  
8189 embedding by a single-layer neural network, applied to a vector of surface features. The  
8190 decision of whether to merge mention  $i$  with entity  $e$  can then be driven by a feedforward  
8191 network,  $\psi_E(i, e) = \text{Feedforward}([v_e; u_i])$ . If  $i$  is added to entity  $e$ , then its representa-  
8192 tion is updated recurrently,  $v_e \leftarrow f(v_e, u_i)$ , using a recurrent neural network such as a  
8193 long short-term memory (LSTM; chapter 6). Alternatively, we can apply a **pooling** oper-  
8194 ation, such as max-pooling or average-pooling (chapter 3), setting  $v_e \leftarrow \text{Pool}(v_e, u_i)$ . In  
8195 either case, the update to the representation of entity  $e$  can be thought of as adding new  
8196 information about the entity from mention  $i$ .

8197 **15.4 Evaluating coreference resolution**

8198 The state of coreference evaluation is aggravatingly complex. Early attempts at sim-  
8199 ple evaluation metrics were found to under-penalize trivial baselines, such as placing  
8200 each mention in its own cluster, or grouping all mentions into a single cluster. Follow-  
8201 ing Denis and Baldridge (2009), the CoNLL 2011 shared task on coreference (Pradhan  
8202 et al., 2011) formalized the practice of averaging across three different metrics: MUC (Vi-  
8203 lain et al., 1995), B-CUBED (Bagga and Baldwin, 1998a), and CEAf (Luo, 2005). Refer-  
8204 ence implementations of these metrics are available from Pradhan et al. (2014) at <https://github.com/conll/reference-coreference-scorers>.

8206 **Additional resources**

8207 Ng (2010) surveys coreference resolution through 2010. Early work focused exclusively  
8208 on pronoun resolution, with rule-based (Lappin and Leass, 1994) and probabilistic meth-  
8209 ods (Ge et al., 1998). The full coreference resolution problem was popularized in a shared  
8210 task associated with the sixth Message Understanding Conference, which included coref-  
8211 erence annotations for training and test sets of thirty documents each (Grishman and  
8212 Sundheim, 1996). An influential early paper was the decision tree approach of Soon et al.  
8213 (2001), who introduced mention ranking. A comprehensive list of surface features for  
8214 coreference resolution is offered by Bengtson and Roth (2008). Durrett and Klein (2013)  
8215 improved on prior work by introducing a large lexicalized feature set; subsequent work  
8216 has emphasized neural representations of entities and mentions (Wiseman et al., 2015).

8217 **Exercises**

- 8218 1. Select an article from today's news, and annotate coreference for the first twenty  
8219 noun phrases that appear in the article (include nested noun phrases). That is,  
8220 group the noun phrases into entities, where each entity corresponds to a set of noun  
8221 phrases. Then specify the mention-pair training data that would result from the first  
8222 five noun phrases.
  - 8223 2. Using your annotations from the preceding problem, compute the following statistics:  
8224
    - 8225 • The number of times new entities are introduced by each of the three types of  
8226 referring expressions: pronouns, proper nouns, and nominals. Include "single-  
8227 ton" entities that are mentioned only once.
    - 8228 • For each type of referring expression, compute the fraction of mentions that are  
8229 anaphoric.
  - 8230 3. Apply a simple heuristic to all pronouns in the article from the previous exercise.  
8231 Specifically, link each pronoun to the closest preceding noun phrase that agrees in  
8232 gender, number, animacy, and person. Compute the following evaluation:  
8233
    - 8234 • True positive: a pronoun that is linked to a noun phrase with which it is coref-  
erent, or is correctly labeled as the first mention of an entity.
    - 8235 • False positive: a pronoun that is linked to a noun phrase with which it is not  
8236 coreferent. (This includes mistakenly linking singleton or non-referential pro-  
8237 nouns.)
    - 8238 • False negative: a pronoun that is not linked to a noun phrase with which it is  
8239 coreferent.
- 8240 Compute the *F-MEASURE* for your method, and for a trivial baseline in which ev-  
8241 ery mention is its own entity. Are there any additional heuristics that would have  
8242 improved the performance of this method?
- 8243 4. Durrett and Klein (2013) compute the probability of the gold coreference clustering  
8244 by summing over all antecedent structures that are compatible with the clustering.  
8245 Compute the number of antecedent structures for a single entity with  $K$  mentions.
  - 8246 5. Use the policy gradient algorithm to compute the gradient for the following sce-  
8247 nario, based on the Bell tree in Figure 15.4:
    - 8248 • The gold clustering  $c^*$  is  $\{Abigail, her\}, \{she\}$ .

- Drawing a single sequence of actions ( $K = 1$ ) from the current policy, you obtain the following incremental clusterings:

$$\begin{aligned}\mathbf{c}(a_1) &= \{\text{Abigail}\} \\ \mathbf{c}(\mathbf{a}_{1:2}) &= \{\text{Abigail}, \text{she}\} \\ \mathbf{c}(\mathbf{a}_{1:3}) &= \{\text{Abigail}, \text{she}\}, \{\text{her}\}.\end{aligned}$$

- 8249     • At each mention  $t$ , the action space  $\mathcal{A}_t$  is to merge the mention with each exist-  
8250     ing cluster, or the empty cluster, with probability,

$$\Pr(\text{Merge}(m_t, \mathbf{c}(\mathbf{a}_{1:t-1}))) \propto \exp \psi_E(m_t \cup \mathbf{c}(\mathbf{a}_{1:t-1})), \quad [15.34]$$

8251 where the cluster score  $\psi_E(m_t \cup c)$  is defined in Equation 15.15.

8252 Compute the gradient  $\frac{\partial}{\partial \theta} L(\theta)$  in terms of the loss  $\ell(c(a))$  and the features of each  
8253 (potential) cluster. Explain the differences between the gradient-based update  $\theta \leftarrow \theta - \frac{\partial}{\partial \theta} L(\theta)$   
8254 and the incremental perceptron update from this sample example.



8255 **Chapter 16**

8256 **Discourse**

8257 Applications of natural language processing often concern multi-sentence documents:  
8258 from paragraph-long restaurant reviews, to 500-word newspaper articles, to 500-page  
8259 novels. Yet most of the methods that we have discussed thus far are concerned with  
8260 individual sentences. This chapter discusses theories and methods for handling multi-  
8261 sentence linguistic phenomena, known collectively as **discourse**. There are diverse char-  
8262 acterizations of discourse structure, and no single structure is ideal for every computa-  
8263 tional application. This chapter covers some of the most well studied discourse repre-  
8264 sentations, while highlighting computational models for identifying and exploiting these  
8265 structures.

8266 **16.1 Segments**

8267 A document or conversation can be viewed as a sequence of **segments**, each of which is  
8268 **cohesive** in its content and/or function. In Wikipedia biographies, these segments often  
8269 pertain to various aspects to the subject's life: early years, major events, impact on others,  
8270 and so on. This segmentation is organized around **topics**. Alternatively, scientific research  
8271 articles are often organized by **functional themes**: the introduction, a survey of previous  
8272 research, experimental setup, and results.

8273 Written texts often mark segments with section headers and related formatting de-  
8274 vices. However, such formatting may be too coarse-grained to support applications such  
8275 as the retrieval of specific passages of text that are relevant to a query (Hearst, 1997).  
8276 Unformatted speech transcripts, such as meetings and lectures, are also an application  
8277 scenario for segmentation (Carletta, 2007; Glass et al., 2007; Janin et al., 2003).

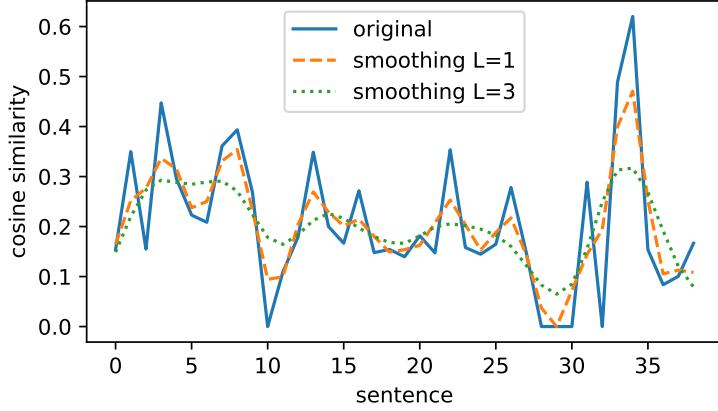


Figure 16.1: Smoothed cosine similarity among adjacent sentences in a news article. Local minima at  $m = 10$  and  $m = 29$  indicate likely segmentation points.

### 8278 16.1.1 Topic segmentation

A cohesive topic segment forms a unified whole, using various linguistic devices: repeated references to an entity or event; the use of conjunctions to link related ideas; and the repetition of meaning through lexical choices (Halliday and Hasan, 1976). Each of these cohesive devices can be measured, and then used as features for topic segmentation. A classical example is the use of lexical cohesion in the `TextTiling` method for topic segmentation (Hearst, 1997). The basic idea is to compute the textual similarity between each pair of adjacent blocks of text (sentences or fixed-length units), using a formula such as the smoothed **cosine similarity** of their bag-of-words vectors,

$$s_m = \frac{\mathbf{x}_m \cdot \mathbf{x}_{m+1}}{\|\mathbf{x}_m\|_2 \times \|\mathbf{x}_{m+1}\|_2} \quad [16.1]$$

$$\bar{s}_m = \sum_{\ell=0}^L k_\ell (s_{m+\ell} + s_{m-\ell}), \quad [16.2]$$

8279 with  $k_\ell$  representing the value of a smoothing kernel of size  $L$ , e.g.  $\mathbf{k} = [1, 0.5, 0.25]^\top$ .  
 8280 Segmentation points are then identified at local minima in the smoothed similarities  $\bar{s}$ ,  
 8281 since these points indicate changes in the overall distribution of words in the text. An  
 8282 example is shown in Figure 16.1.

8283 Text segmentation can also be formulated as a probabilistic model, in which each seg-  
 8284 ment has a unique language model that defines the probability over the text in the seg-  
 8285 ment (Utiyama and Isahara, 2001; Eisenstein and Barzilay, 2008; Du et al., 2013).<sup>1</sup> A good

---

<sup>1</sup>There is a rich literature on how latent variable models (such as **latent Dirichlet allocation**) can track

8286 segmentation achieves high likelihood by grouping segments with similar word distribu-  
8287 tions. This probabilistic approach can be extended to **hierarchical topic segmentation**, in  
8288 which each topic segment is divided into subsegments (Eisenstein, 2009). All of these ap-  
8289 proaches are unsupervised. While labeled data can be obtained from well-formatted texts  
8290 such as textbooks, such annotations may not generalize to speech transcripts in alterna-  
8291 tive domains. Supervised methods have been tried in cases where in-domain labeled data  
8292 is available, substantially improving performance by learning weights on multiple types  
8293 of features (Galley et al., 2003).

### 8294 16.1.2 Functional segmentation

8295 In some genres, there is a canonical set of communicative *functions*: for example, in sci-  
8296 entific research articles, one such function is to communicate the general background for  
8297 the article, another is to introduce a new contribution, or to describe the aim of the re-  
8298 search (Teufel et al., 1999). A **functional segmentation** divides the document into con-  
8299 tiguous segments, sometimes called **rhetorical zones**, in which each sentence has the same  
8300 function. Teufel and Moens (2002) train a supervised classifier to identify the functional  
8301 of each sentence in a set of scientific research articles, using features that describe the sen-  
8302 tence's position in the text, its similarity to the rest of the article and title, tense and voice of  
8303 the main verb, and the functional role of the previous sentence. Functional segmentation  
8304 can also be performed without supervision. Noting that some types of Wikipedia arti-  
8305 cles have very consistent functional segmentations (e.g., articles about cities or chemical  
8306 elements), Chen et al. (2009) introduce an unsupervised model for functional segmenta-  
8307 tion, which learns both the language model associated with each function and the typical  
8308 patterning of functional segments across the article.

## 8309 16.2 Entities and reference

8310 Another dimension of discourse relates to which entities are mentioned throughout the  
8311 text, and how. Consider the examples in Figure 16.2: Grosz et al. (1995) argue that the first  
8312 discourse is more coherent. Do you agree? The examples differ in their choice of **refe-  
8313 ring expressions** for the protagonist *John*, and in the syntactic constructions in sentences  
8314 (b) and (d). The examples demonstrate the need for theoretical models to explain how  
8315 referring expressions are chosen, and where they are placed within sentences. Such mod-  
8316 els can then be used to help interpret the overall structure of the discourse, to measure  
8317 discourse coherence, and to generate discourses in which referring expressions are used  
8318 coherently.

---

topics across documents (Blei et al., 2003; Blei, 2012).

- |  |   |
|--|---|
| (16.1) a. John went to his favorite music store to buy a piano.<br>b. He had frequented the store for many years.<br>c. He was excited that he could finally buy a piano.<br>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.<br>b. It was a store John had frequented for many years.<br>c. He was excited that he could finally buy a piano.<br>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.

### 8319 16.2.1 Centering theory

8320 The relationship between discourse and entity reference is most elaborated in **centering**  
 8321 **theory** (Grosz et al., 1995). According to the theory, every utterance in the discourse is  
 8322 characterized by a set of entities, known as *centers*.

- 8323 • The **forward-looking centers** in utterance  $m$  are all the entities that are mentioned  
 8324 in the utterance,  $c_f(w_m) = \{e_1, e_2, \dots\}$ . The forward-looking centers are partially  
 8325 ordered by their syntactic prominence, favoring subjects over other positions.  
 8326 • The **backward-looking center**  $c_b(w_m)$  is the highest-ranked element in the set of  
 8327 forward-looking centers from the previous utterance  $c_f(w_{m-1})$  that is also men-  
 8328 tioned in  $w_m$ .

8329 Given these two definitions, centering theory makes the following predictions about  
 8330 the form and position of referring expressions:

- 8331 1. If a pronoun appears in the utterance  $w_m$ , then the backward-looking center  $c_b(w_m)$   
 8332 must also be realized as a pronoun. This rule argues against the use of *it* to refer  
 8333 to the piano store in Example (16.2d), since JOHN is the backward looking center of  
 8334 (16.2d), and he is mentioned by name and not by a pronoun.  
 8335 2. Sequences of utterances should retain the same backward-looking center if possible,  
 8336 and ideally, the backward-looking center should also be the top-ranked element in  
 8337 the list of forward-looking centers. This rule argues in favor of the preservation of  
 8338 JOHN as the backward-looking center throughout Example (16.1).

8339 Centering theory unifies aspects of syntax, discourse, and anaphora resolution. However,  
 8340 it can be difficult to clarify exactly how to rank the elements of each utterance, or even  
 8341 how to partition a text or dialog into utterances (Poesio et al., 2004).

	SKYLER	WALTER	DANGER	A GUY	THE DOOR
<i>You don't know who you're talking to,</i>	S	-	-	-	-
<i>so let me clue you in.</i>	O	O	-	-	-
<i>I am not in danger, Skyler.</i>	X	S	X	-	-
<i>I am the danger.</i>	-	S	O	-	-
<i>A guy opens his door and gets shot,</i>	-	-	-	S	O
<i>and you think that of me?</i>	S	X	-	-	-
<i>No. I am the one who knocks!</i>	-	S	-	-	-

Figure 16.3: The entity grid representation for a dialogue from the television show *Breaking Bad*.

### 16.2.2 The entity grid

One way to formalize the ideas of centering theory is to arrange the entities in a text or conversation in an **entity grid**. This is a data structure with one row per sentence, and one column per entity (Barzilay and Lapata, 2008). Each cell  $c(m, i)$  can take the following values:

$$c(m, i) = \begin{cases} S, & \text{entity } i \text{ is in subject position in sentence } m \\ O, & \text{entity } i \text{ is in object position in sentence } m \\ X, & \text{entity } i \text{ appears in sentence } m, \text{ in neither subject nor object position} \\ -, & \text{entity } i \text{ does not appear in sentence } m. \end{cases} \quad [16.3]$$

To populate the entity grid, syntactic parsing is applied to identify subject and object positions, and coreference resolution is applied to link multiple mentions of a single entity. An example is shown in Figure 16.3.

After the grid is constructed, the coherence of a document can be measured by the transitions between adjacent cells in each column. For example, the transition  $(S \rightarrow S)$  keeps an entity in subject position across adjacent sentences; the transition  $(O \rightarrow S)$  promotes an entity from object position to subject position; the transition  $(S \rightarrow -)$  drops the subject of one sentence from the next sentence. The probabilities of each transition can be estimated from labeled data, and an entity grid can then be scored by the sum of the log-probabilities across all columns and all transitions,  $\sum_{i=1}^{N_e} \sum_{m=1}^M \log p(c(m, i) | c(m-1, i))$ . The resulting probability can be used as a proxy for the coherence of a text. This has been shown to be useful for a range of tasks: determining which of a pair of articles is more readable (Schwartz and Ostendorf, 2005), correctly ordering the sentences in a scrambled

8360 text (Lapata, 2003), and disentangling multiple conversational threads in an online multi-  
 8361 party chat (Elsner and Charniak, 2010).

8362 **16.2.3 \*Formal semantics beyond the sentence level**

8363 An alternative view of the role of entities in discourse focuses on formal semantics, and the  
 8364 construction of meaning representations for multi-sentence units. Consider the following  
 8365 two sentences (from Bird et al., 2009):

- 8366 (16.3) a. Angus owns a dog.  
 8367 b. It bit Irene.

8368 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:

$$\exists x. \text{DOG}(x) \wedge \text{OWN}(\text{ANGUS}, x) \quad [16.5]$$

$$\text{BITE}(y, \text{IRENE}). \quad [16.6]$$

8369 Unifying these two representations into the form of Equation 16.4 requires linking the  
 8370 unbound variable  $y$  from [16.6] with the quantified variable  $x$  in [16.5]. Discourse under-  
 8371 standing therefore requires the reader to update a set of assignments, from variables  
 8372 to entities. This update would (presumably) link the *dog* in the first sentence of [16.3]  
 8373 with the unbound variable  $y$  in the second sentence, thereby licensing the conjunction in  
 8374 [16.4].<sup>2</sup> This basic idea is at the root of **dynamic semantics** (Groenendijk and Stokhof,  
 8375 1991). **Segmented discourse representation theory** links dynamic semantics with a set  
 8376 of **discourse relations**, which explain how adjacent units of text are rhetorically or con-  
 8377 ceptually related (Lascarides and Asher, 2007). The next section explores the theory of  
 8378 discourse relations in more detail.

8379 **16.3 Relations**

8380 In dependency grammar, sentences are characterized by a graph (usually a tree) of syntac-  
 8381 tic relations between words, such as NSUBJ and DET. A similar idea can be applied at the  
 8382 document level, identifying relations between discourse units, such as clauses, sentences,  
 8383 or paragraphs. The task of **discourse parsing** involves identifying discourse units and  
 8384 the relations that hold between them. These relations can then be applied to tasks such as  
 8385 document classification and summarization, as discussed in § 16.3.4.

---

<sup>2</sup>This 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.

### 8386 16.3.1 Shallow discourse relations

8387 The existence of discourse relations is hinted by **discourse connectives**, such as *however*,  
 8388 *moreover*, *meanwhile*, and *if ... then*. These connectives explicitly specify the relationship  
 8389 between adjacent units of text: *however* signals a contrastive relationship, *moreover* signals  
 8390 that the subsequent text elaborates or strengthens the point that was made immediately  
 8391 beforehand, *meanwhile* indicates that two events are contemporaneous, and *if ... then* sets  
 8392 up a conditional relationship. Discourse connectives can therefore be viewed as a starting  
 8393 point for the analysis of discourse relations.

8394 In **lexicalized tree-adjoining grammar for discourse (D-LTAG)**, each connective an-  
 8395 chors a relationship between two units of text (Webber, 2004). This model provides the  
 8396 theoretical basis for the **Penn Discourse Treebank (PDTB)**, the largest corpus of discourse  
 8397 relations in English (Prasad et al., 2008). It includes a hierarchical inventory of discourse  
 8398 relations (shown in Table 16.1), which is created by abstracting the meanings implied by  
 8399 the discourse connectives that appear in real texts (Knott, 1996). These relations are then  
 8400 annotated on the same corpus of news text used in the Penn Treebank (see § 9.2.2), adding  
 8401 the following information:

- Each connective is annotated for the discourse relation or relations that it expresses, if any — many discourse connectives have senses in which they do not signal a discourse relation (Pitler and Nenkova, 2009).
- For each discourse relation, the two arguments of the relation are specified as ARG1 and ARG2, where ARG2 is constrained to be adjacent to the connective. These arguments may be sentences, but they may also smaller or larger units of text.
- Adjacent sentences are annotated for **implicit discourse relations**, which are not marked by any connective. When a connective could be inserted between a pair of sentence, the annotator supplies it, and also labels its sense (e.g., example 16.5). In some cases, there is no relationship at all between a pair of adjacent sentences; in other cases, the only relation is that the adjacent sentences mention one or more shared entity. These phenomena are annotated as NOREL and ENTRREL (entity relation), respectively.

Examples of Penn Discourse Treebank annotations are shown in (16.4). In (16.4), the word *therefore* acts as an explicit discourse connective, linking the two adjacent units of text. The Treebank annotations also specify the “sense” of each relation, linking the connective to a relation in the sense inventory shown in Table 16.1: in (16.4), the relation is PRAGMATIC CAUSE:JUSTIFICATION because it relates to the author’s communicative intentions. The word *therefore* can also signal causes in the external world (e.g., *He was therefore forced to relinquish his plan*). In **discourse sense classification**, the goal is to determine which discourse relation, if any, is expressed by each connective. A related task is the classification of implicit discourse relations, as in (16.5). In this example, the relationship between the adjacent sentences could be expressed by the connective *because*, indicating a CAUSE:REASON relationship.

#### 16.3.1.1 Classifying explicit discourse relations and their arguments

As suggested by the examples above, many connectives can be used to invoke multiple types of discourse relations. Similarly, some connectives have senses that are unrelated to discourse: for example, *and* functions as a discourse connective when it links propositions, but not when it links noun phrases (Lin et al., 2014). Nonetheless, the senses of explicitly-marked discourse relations in the Penn Treebank are relatively easy to classify, at least at the coarse-grained level. When classifying the four top-level PDTB relations, 90% accuracy can be obtained simply by selecting the most common relation for each connective (Pitler and Nenkova, 2009). At the more fine-grained levels of the discourse relation hierarchy, connectives are more ambiguous. This fact is reflected both in the accuracy of automatic sense classification (Versley, 2011) and in interannotator agreement, which falls to 80% for level-3 discourse relations (Prasad et al., 2008).

- (16.4) *...as this business of whaling has somehow come to be regarded among landsmen as a rather unpoetical and disreputable pursuit; therefore, I am all anxiety to convince ye, ye landsmen, of the injustice hereby done to us hunters of whales.*
- (16.5) But a few funds have taken other defensive steps. *Some have raised their cash positions to record levels. Implicit = BECAUSE High cash positions help buffer a fund when the market falls.*
- (16.6) Michelle lives in a hotel room, and although **she drives a canary-colored Porsche**, *she hasn't time to clean or repair it.*
- (16.7) *Most oil companies, when they set exploration and production budgets for this year, forecast revenue of \$15 for each barrel of crude produced.*

Figure 16.4: Example annotations of discourse relations. In the style of the Penn Discourse Treebank, the discourse connective is underlined, the first argument is shown in italics, and the second argument is shown in bold. Examples (16.5-16.7) are quoted from Prasad et al. (2008).

8438 A more challenging task for explicitly-marked discourse relations is to identify the  
 8439 scope of the arguments. Discourse connectives need not be adjacent to ARG1, as shown  
 8440 in item 16.6, where ARG1 follows ARG2; furthermore, the arguments need not be contigu-  
 8441 ous, as shown in (16.7). For these reasons, recovering the arguments of each discourse  
 8442 connective is a challenging subtask. Because intra-sentential arguments are often syn-  
 8443 tactic constituents (see chapter 10), many approaches train a classifier to predict whether  
 8444 each constituent is an appropriate argument for each explicit discourse connective (Well-  
 8445 ner and Pustejovsky, 2007; Lin et al., 2014, e.g.,).

#### 8446 16.3.1.2 Classifying implicit discourse relations

Implicit discourse relations are considerably more difficult to classify and to annotate.<sup>3</sup> Most approaches are based on an encoding of each argument, which is then used as input to a non-linear classifier:

$$\mathbf{z}^{(i)} = \text{Encode}(\mathbf{w}^{(i)}) \quad [16.7]$$

$$\mathbf{z}^{(i+1)} = \text{Encode}(\mathbf{w}^{(i+1)}) \quad [16.8]$$

$$\hat{y}_i = \underset{y}{\operatorname{argmax}} \Psi(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}). \quad [16.9]$$

---

<sup>3</sup>In 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).

8447 This basic framework can be instantiated in several ways, including both feature-based  
 8448 and neural encoders. Several recent approaches are compared in the 2015 and 2016 shared  
 8449 tasks at the Conference on Natural Language Learning (Xue et al., 2015, 2016).

8450 **Feature-based approaches** Each argument can be encoded into a vector of surface fea-  
 8451 tures. The encoding typically includes lexical features (all words, or all content words, or  
 8452 a subset of words such as the first three and the main verb), Brown clusters of individ-  
 8453 ual words (§ 14.4), and syntactic features such as terminal productions and dependency  
 8454 arcs (Pitler et al., 2009; Lin et al., 2009; Rutherford and Xue, 2014). The classification func-  
 8455 tion then has two parts. First, it creates a joint feature vector by combining the encodings  
 8456 of each argument, typically by computing the cross-product of all features in each encod-  
 8457 ing:

$$\mathbf{f}(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}) = \{(a \times b \times y) : (\mathbf{z}_a^{(i)} \mathbf{z}_b^{(i+1)})\} \quad [16.10]$$

8458 The size of this feature set grows with the square of the size of the vocabulary, so it can be  
 8459 helpful to select a subset of features that are especially useful on the training data (Park  
 8460 and Cardie, 2012). After  $\mathbf{f}$  is computed, any classifier can be trained to compute the final  
 8461 score,  $\Psi(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}) = \boldsymbol{\theta} \cdot \mathbf{f}(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)})$ .

8462 **Neural network approaches** In neural network architectures, the encoder is learned  
 8463 jointly with the classifier as an end-to-end model. Each argument can be encoded using  
 8464 a variety of neural architectures (surveyed in § 14.8): recursive (§ 10.6.1; Ji and Eisenstein,  
 8465 2015), recurrent (§ 6.3; Ji et al., 2016), and convolutional (§ 3.4; Qin et al., 2017). The clas-  
 8466 sification function can then be implemented as a feedforward neural network on the two  
 8467 encodings (chapter 3; for examples, see Rutherford et al., 2017; Qin et al., 2017), or as a  
 8468 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  
 8469 encoding model can be trained by backpropagation from the classification objective, such  
 8470 as the margin loss. Rutherford et al. (2017) show that neural architectures outperform  
 8471 feature-based approaches in most settings. While neural approaches require engineering  
 8472 the network architecture (e.g., embedding size, number of hidden units in the classifier),  
 8473 feature-based approaches also require significant engineering to incorporate linguistic re-  
 8474 sources such as Brown clusters and parse trees, and to select a subset of relevant features.

### 8475 16.3.2 Hierarchical discourse relations

8476 In sentence parsing, adjacent phrases combine into larger constituents, ultimately pro-  
 8477 ducing a single constituent for the entire sentence. The resulting tree structure enables  
 8478 structured analysis of the sentence, with subtrees that represent syntactically coherent  
 8479 chunks of meaning. **Rhetorical Structure Theory (RST)** extends this style of hierarchical  
 8480 analysis to the discourse level (Mann and Thompson, 1988).

8481     The basic element of RST is the **discourse unit**, which refers to a contiguous span of  
 8482     text. **Elementary discourse units** (EDUs) are the atomic elements in this framework, and  
 8483     are typically (but not always) clauses.<sup>4</sup> Each discourse relation combines two or more  
 8484     adjacent discourse units into a larger, composite discourse unit; this process ultimately  
 8485     unites the entire text into a tree-like structure.<sup>5</sup>

8486 **Nuclearity** In many discourse relations, one argument is primary. For example:

8487 (16.8) [LaShawn loves animals]<sub>N</sub>  
 8488        [She has nine dogs and one pig]<sub>S</sub>

8489 In this example, the second sentence provides EVIDENCE for the point made in the first  
 8490 sentence. The first sentence is thus the **nucleus** of the discourse relation, and the second  
 8491 sentence is the **satellite**. The notion of **nuclearity** is analogous to the head-modifier struc-  
 8492 ture of dependency parsing (see § 11.1.1). However, in RST, some relations have multiple  
 8493 nuclei. For example, the arguments of the CONTRAST relation are equally important:

8494 (16.9) [The clash of ideologies survives this treatment]<sub>N</sub>  
 8495        [but the nuance and richness of Gorky's individual characters have vanished in the scuffle]<sub>N</sub><sup>6</sup>

8496 Relations that have multiple nuclei are called **coordinating**; relations with a single nu-  
 8497 cleus are called **subordinating**. Subordinating relations are constrained to have only two  
 8498 arguments, while coordinating relations (such as CONJUNCTION) may have more than  
 8499 two.

8500 **RST Relations** Rhetorical structure theory features a large inventory of discourse rela-  
 8501 tions, which are divided into two high-level groups: subject matter relations, and presen-  
 8502 tational relations. Presentational relations are organized around the intended beliefs of  
 8503 the reader. For example, in (16.8), the second discourse unit provides evidence intended  
 8504 to increase the reader's belief in the proposition expressed by the first discourse unit, that  
 8505 *LaShawn loves animals*. In contrast, subject-matter relations are meant to communicate ad-  
 8506 dditional facts about the propositions contained in the discourse units that they relate:

---

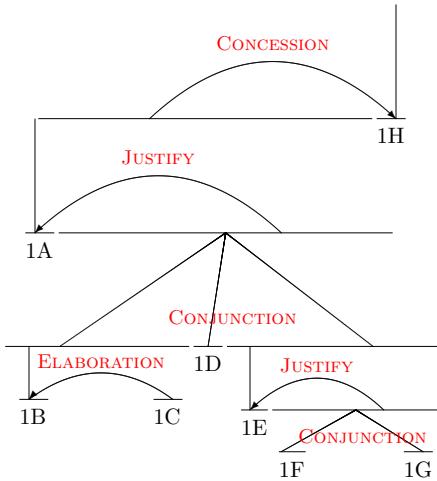
<sup>4</sup>Details of discourse segmentation can be found in the RST annotation manual (Carlson and Marcu, 2001).

<sup>5</sup>While 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.

<sup>6</sup>from the RST Treebank (Carlson et al., 2002)



[It could have been a great movie]<sup>1A</sup> [It does have beautiful scenery,]<sup>1B</sup> [some of the best since Lord of the Rings.]<sup>1C</sup> [The acting is well done,]<sup>1D</sup> [and I really liked the son of the leader of the Samurai.]<sup>1E</sup> [He was a likable chap,]<sup>1F</sup> [and I hated to see him die.]<sup>1G</sup> [But, other than all that, this movie is nothing more than hidden rip-offs.]<sup>1H</sup>

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.

8507 (16.10) [the debt plan was rushed to completion]<sub>N</sub>  
 8508 [in order to be announced at the meeting]<sub>S</sub><sup>7</sup>

8509 In this example, the satellite describes a world state that is realized by the action described  
 8510 in the nucleus. This relationship is about the world, and not about the author's commu-  
 8511 nicative intentions.

8512 **Example** Figure 16.5 depicts an RST analysis of a paragraph from a movie review. Asym-  
 8513 metric (subordinating) relations are depicted with an arrow from the satellite to the nu-  
 8514 cleus; symmetric (coordinating) relations are depicted with lines. The elementary dis-  
 8515 course units 1F and 1G are combined into a larger discourse unit with the symmetric  
 8516 CONJUNCTION relation. The resulting discourse unit is then the satellite in a JUSTIFY  
 8517 relation with 1E.

<sup>7</sup>from the RST Treebank (Carlson et al., 2002)

8518 **16.3.2.1 Hierarchical discourse parsing**

8519 The goal of discourse parsing is to recover a hierarchical structural analysis from a doc-  
 8520 ument text, such as the analysis in Figure 16.5. For now, let's assume a segmentation  
 8521 of the document into elementary discourse units (EDUs); segmentation algorithms are  
 8522 discussed below. After segmentation, discourse parsing can be viewed as a combination  
 8523 of two components: the discourse relation classification techniques discussed in § 16.3.1.2,  
 8524 and algorithms for phrase-structure parsing, such as chart parsing and shift-reduce, which  
 8525 were discussed in chapter 10.

8526 Both chart parsing and shift-reduce require encoding composite discourse units, ei-  
 8527 ther in a discrete feature vector or a dense neural representation.<sup>8</sup> Some discourse parsers  
 8528 rely on the **strong compositionality criterion** (Marcu, 1996), which states the assumption  
 8529 that a composite discourse unit can be represented by its nucleus. This criterion is used in  
 8530 feature-based discourse parsing to determine the feature vector for a composite discourse  
 8531 unit (Hernault et al., 2010); it is used in neural approaches to setting the vector encod-  
 8532 ing for a composite discourse unit equal to the encoding of its nucleus (Ji and Eisenstein,  
 8533 2014). An alternative neural approach is to learn a composition function over the compo-  
 8534 nents of a composite discourse unit (Li et al., 2014), using a recursive neural network (see  
 8535 § 14.8.3).

8536 **Bottom-up discourse parsing** Assume a segmentation of the text into  $N$  elementary  
 8537 discourse units with base representations  $\{z^{(i)}\}_{i=1}^N$ , and assume a composition function  
 8538  $\text{COMPOSE}(z^{(i)}, z^{(j)}, \ell)$ , which maps two encodings and a discourse relation  $\ell$  into a new  
 8539 encoding. The composition function can follow the strong compositionality criterion and  
 8540 simply select the encoding of the nucleus, or it can do something more complex. We  
 8541 also need a scoring function  $\Psi(z^{(i,k)}, z^{(k,j)}, \ell)$ , which computes a scalar score for the (bi-  
 8542 narized) discourse relation  $\ell$  with left child covering the span  $i + 1 : k$ , and the right  
 8543 child covering the span  $k + 1 : j$ . Given these components, we can construct vector rep-  
 8544 resentations for each span, and this is the basic idea underlying **compositional vector**  
 8545 **grammars** (Socher et al., 2013).

8546 These same components can also be used in bottom-up parsing, in a manner that is  
 8547 similar to the CKY algorithm for weighted context-free grammars (see § 10.1): compute  
 8548 the score and best analysis for each possible span of increasing lengths, while storing  
 8549 back-pointers that make it possible to recover the optimal parse of the entire input. How-  
 8550 ever, there is an important distinction from CKY parsing: for each labeled span  $(i, j, \ell)$ , we  
 8551 must use the composition function to construct a representation  $z^{(i,j,\ell)}$ . This representa-  
 8552 tion is then used to combine the discourse unit spanning  $i + 1 : j$  in higher-level discourse  
 8553 relations. The representation  $z^{(i,j,\ell)}$  depends on the entire substructure of the unit span-

---

<sup>8</sup>To 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).

8554 ning  $i + 1 : j$ , and this violates the locality assumption that underlie CKY’s optimality  
 8555 guarantee. Bottom-up parsing with recursively constructed span representations is gen-  
 8556 erally not guaranteed to find the best-scoring discourse parse. This problem is explored  
 8557 in an exercise at the end of the chapter.

8558 **Transition-based discourse parsing** One drawback of bottom-up parsing is its cubic  
 8559 time complexity in the length of the input. For long documents, transition-based parsing  
 8560 is an appealing alternative. The shift-reduce algorithm can be applied to discourse parsing  
 8561 fairly directly (Sagae, 2009): the stack stores a set of discourse units and their repres-  
 8562 entations, and each action is chosen by a function of these representations. This function  
 8563 could be a linear product of weights and features, or it could be a neural network ap-  
 8564 plied to encodings of the discourse units. The REDUCE action then performs composition  
 8565 on the two discourse units at the top of the stack, yielding a larger composite discourse  
 8566 unit, which goes on top of the stack. All of the techniques for integrating learning and  
 8567 transition-based parsing, described in § 11.3, are applicable to discourse parsing.

### 8568 16.3.2.2 Segmenting discourse units

8569 In rhetorical structure theory, elementary discourse units do not cross the sentence bound-  
 8570 ary, so discourse segmentation can be performed within sentences, assuming the sentence  
 8571 segmentation is given. The segmentation of sentences into elementary discourse units is  
 8572 typically performed using features of the syntactic analysis (Braud et al., 2017). One ap-  
 8573 proach is to train a classifier to determine whether each syntactic constituent is an EDU,  
 8574 using features such as the production, tree structure, and head words (Soricut and Marcu,  
 8575 2003; Hernault et al., 2010). Another approach is to train a sequence labeling model, such  
 8576 as a conditional random field (Sporleder and Lapata, 2005; Xuan Bach et al., 2012; Feng  
 8577 et al., 2014). This is done using the BIO formalism for segmentation by sequence labeling,  
 8578 described in § 8.3.

### 8579 16.3.3 Argumentation

8580 An alternative view of text-level relational structure focuses on **argumentation** (Stab and  
 8581 Gurevych, 2014b). Each segment (typically a sentence or clause) may support or rebut  
 8582 another segment, creating a graph structure over the text. In the following example (from  
 8583 Peldszus and Stede, 2013), segment  $S_2$  provides argumentative support for the proposi-  
 8584 tion in the segment  $S_1$ :

8585 (16.11) [We should tear the building down] $_{S1}$   
 8586 [because it is full of asbestos] $_{S2}$ .

8587 Assertions may also support or rebut proposed links between two other assertions, cre-  
 8588 ating a **hypergraph**, which is a generalization of a graph to the case in which edges can

8589 join any number of vertices. This can be seen by introducing another sentence into the  
8590 example:

8591 (16.12) [In principle it is possible to clean it up.]<sub>S3</sub>  
8592 [but according to the mayor that is too expensive.]<sub>S4</sub>

8593 S3 acknowledges the validity of *S2*, but undercuts its support of *S1*. This can be repre-  
8594 sented by introducing a hyperedge,  $(S3, S2, S1)_{\text{undercut}}$ , indicating that *S3* undercuts the  
8595 proposed relationship between *S2* and *S1*. *S4* then undercuts the relevance of *S3*.

8596 **Argumentation mining** is the task of recovering such structures from raw texts. At  
8597 present, annotations of argumentation structure are relatively small: Stab and Gurevych  
8598 (2014a) have annotated a collection of 90 persuasive essays, and Peldszus and Stede (2015)  
8599 have solicited and annotated a set of 112 paragraph-length “microtexts” in German.

#### 8600 16.3.4 Applications of discourse relations

8601 The predominant application of discourse parsing is to select content within a document.  
8602 In rhetorical structure theory, the nucleus is considered the more important element of  
8603 the relation, and is more likely to be part of a summary of the document; it may also  
8604 be more informative for document classification. The D-LTAG theory that underlies the  
8605 Penn Discourse Treebank lacks this notion of nuclearity, but arguments may have varying  
8606 importance, depending on the relation type. For example, the span of text constituting  
8607 ARG1 of an expansion relation is more likely to appear in a summary, while the sentence  
8608 constituting ARG2 of an implicit relation is less likely (Louis et al., 2010). Discourse rela-  
8609 tions may also signal segmentation points in the document structure. Explicit discourse  
8610 markers have been shown to correlate with changes in subjectivity, and identifying such  
8611 change points can improve document-level sentiment classification, by helping the clas-  
8612 sifier to focus on the subjective parts of the text (Trivedi and Eisenstein, 2013; Yang and  
8613 Cardie, 2014).

##### 8614 16.3.4.1 Extractive Summarization

8615 Text **summarization** is the problem of converting a longer text into a shorter one, while  
8616 still conveying the key facts, events, ideas, and sentiments from the original. In **extractive**  
8617 **summarization**, the summary is a subset of the original text; in **abstractive summariza-**  
8618 **tion**, the summary is produced *de novo*, by paraphrasing the original, or by first encoding  
8619 it into a semantic representation (see § 19.2). The main strategy for extractive summa-  
8620 rization is to maximize **coverage**, choosing a subset of the document that best covers the  
8621 concepts mentioned in the document as a whole; typically, coverage is approximated by  
8622 bag-of-words overlap (Nenkova and McKeown, 2012). Coverage-based objectives can be  
8623 supplemented by hierarchical discourse relations, using the principle of nuclearity: in any  
8624 subordinating discourse relation, the nucleus is more critical to the overall meaning of the

8625 text, and is therefore more important to include in an extractive summary (Marcu, 1997a).<sup>9</sup>  
 8626 This insight can be generalized from individual relations using the concept of **discourse**  
 8627 **depth** (Hirao et al., 2013): for each elementary discourse unit  $e$ , the discourse depth  $d_e$  is  
 8628 the number of relations in which a discourse unit containing  $e$  is the satellite.

8629 Both discourse depth and nuclearity can be incorporated into extractive summarization  
 8630 using constrained optimization. Let  $\mathbf{x}_n$  be a bag-of-words vector representation of  
 8631 elementary discourse unit  $n$ , let  $y_n \in \{0, 1\}$  indicate whether  $n$  is included in the summary,  
 8632 and let  $d_n$  be the depth of unit  $n$ . Furthermore, let each discourse unit have a “head”  $h$ ,  
 8633 which is defined recursively:

- 8634 • if a discourse unit is produced by a subordinating relation, then its head is the head  
 8635 of the (unique) nucleus;
- 8636 • if a discourse unit is produced by a coordinating relation, then its head is the head  
 8637 of the left-most nucleus;
- 8638 • for each elementary discourse unit, its parent  $\pi(n) \in \{\emptyset, 1, 2, \dots, N\}$  is the head of  
 8639 the smallest discourse unit containing  $n$  whose head is not  $n$ ;
- 8640 • 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),

$$\begin{aligned} & \max_{\mathbf{y}=\{0,1\}^N} \sum_{n=1}^N y_n \frac{\Psi(\mathbf{x}_n, \{\mathbf{x}_{1:N}\})}{d_n} \\ & \text{s.t. } \sum_{n=1}^N y_n \left( \sum_{j=1}^V x_{n,j} \right) \leq L \\ & \quad y_{\pi(n)} \geq y_n, \quad \forall n \end{aligned} \tag{16.11}$$

8641 where  $\Psi(\mathbf{x}_n, \{\mathbf{x}_{1:N}\})$  measures the coverage of elementary discourse unit  $n$  with respect  
 8642 to the rest of the document, and  $\sum_{j=1}^V x_{n,m}$  is the number of tokens in  $\mathbf{x}_n$ . The first con-  
 8643 straint ensures that the number of tokens in the summary has an upper bound  $L$ . The  
 8644 second constraint ensures that no elementary discourse unit is included unless its parent  
 8645 is also included. In this way, the discourse structure is used twice: to downweight the  
 8646 contributions of elementary discourse units that are not central to the discourse, and to  
 8647 ensure that the resulting structure is a subtree of the original discourse parse. The opti-

---

<sup>9</sup>Conversely, the arguments of a multi-nuclear relation should either both be included in the summary, or both excluded (Durrett et al., 2016).

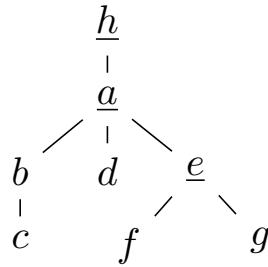


Figure 16.6: A **discourse depth tree** (Hirao et al., 2013) for the discourse parse from Figure 16.5, in which each elementary discourse unit is connected to its parent. The discourse units in one valid summary are underlined.

8648 mization problem in 16.11 can be solved with **integer linear programming**, described in  
 8649 § 13.2.2.<sup>10</sup>

8650 Figure 16.6 shows a **discourse depth tree** for the RST analysis from Figure 16.5, in  
 8651 which each elementary discourse is connected to (and below) its parent. The figure also  
 8652 shows a valid summary, corresponding to:

8653 (16.13) It could have been a great movie, and I really liked the son of the leader of the  
 8654 Samurai. But, other than all that, this movie is nothing more than hidden rip-offs.

#### 8655 16.3.4.2 Document classification

8656 Hierarchical discourse structures lend themselves naturally to text classification: in a sub-  
 8657 ordinating discourse relation, the nucleus should play a stronger role in the classification  
 8658 decision than the satellite. Various implementations of this idea have been proposed.

- 8659 • Focusing on within-sentence discourse relations and lexicon-based classification (see  
 8660 § 4.1.2), Voll and Taboada (2007) simply ignore the text in the satellites of each dis-  
 8661 course relation.
- 8662 • At the document level, elements of each discourse relation argument can be reweighted,  
 8663 favoring words in the nucleus, and disfavoring words in the satellite (Heerschap  
 8664 et al., 2011; Bhatia et al., 2015). This approach can be applied recursively, computing  
 8665 weights across the entire document. The weights can be relation-specific, so that the  
 8666 features from the satellites of contrastive relations are discounted or even reversed.
- 8667 • Alternatively, the hierarchical discourse structure can define the structure of a **re-  
 8668 cursive neural network** (see § 10.6.1). In this network, the representation of each

---

<sup>10</sup>Formally, 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).

8669 discourse unit is computed from its arguments and from a parameter corresponding  
 8670 to the discourse relation (Ji and Smith, 2017).

8671 Shallow, non-hierarchical discourse relations have also been applied to document clas-  
 8672 sification. One approach is to impose a set of constraints on the analyses of individual  
 8673 discourse units, so that adjacent units have the same polarity when they are connected  
 8674 by a discourse relation indicating agreement, and opposite polarity when connected by a  
 8675 contrastive discourse relation, indicating disagreement (Somasundaran et al., 2009; Zirn  
 8676 et al., 2011). Yang and Cardie (2014) apply explicitly-marked relations from the Penn  
 8677 Discourse Treebank to the problem of sentence-level sentiment polarity classification (see  
 8678 § 4.1). They impose the following soft constraints:

- 8679 • When a CONTRAST relation appears between two sentences, those sentences should  
 8680 have opposite sentiment polarity.
- 8681 • When an EXPANSION or CONTINGENCY relation appears between two sentences,  
 8682 they should have the same polarity.
- 8683 • When a CONTRAST relation appears *within* a sentence, it should have neutral polar-  
 8684 ity, since it is likely to express both sentiments.

8685 These discourse-driven constraints are shown to improve performance on two datasets of  
 8686 product reviews.

#### 8687 16.3.4.3 Coherence

8688 Just as **grammaticality** is the property shared by well-structured sentences, **coherence** is  
 8689 the property shared by well-structured discourses. One application of discourse process-  
 8690 ing is to measure (and maximize) the coherence of computer-generated texts like transla-  
 8691 tions and summaries (Kibble and Power, 2004). Coherence assessment is also used to eval-  
 8692 uate human-generated texts, such as student essays (e.g., Miltsakaki and Kukich, 2004;  
 8693 Burstein et al., 2013).

8694 Coherence subsumes a range of phenomena, many of which have been highlighted  
 8695 earlier in this chapter: e.g., that adjacent sentences should be lexically cohesive (Foltz  
 8696 et al., 1998; Ji et al., 2015; Li and Jurafsky, 2017), and that entity references should follow  
 8697 the principles of centering theory (Barzilay and Lapata, 2008; Nguyen and Joty, 2017).  
 8698 Discourse relations also bear on the coherence of a text in a variety of ways:

- 8699 • Hierarchical discourse relations tend to have a “canonical ordering” of the nucleus  
 8700 and satellite (Mann and Thompson, 1988): for example, in the ELABORATION rela-  
 8701 tion from rhetorical structure theory, the nucleus always comes first, while in the  
 8702 JUSTIFICATION relation, the satellite tends to be first (Marcu, 1997b).

- Discourse relations should be signaled by connectives that are appropriate to the semantic or functional relationship between the arguments: for example, a coherent text would be more likely to use *however* to signal a COMPARISON relation than a *temporal* relation (Kibble and Power, 2004).
- Discourse relations tend to appear in predictable sequences: for example, COMPARISON relations tend to immediately precede CONTINGENCY relations (Pitler et al., 2008). This observation can be formalized by generalizing the entity grid model (§ 16.2.2), so that each cell  $(i, j)$  provides information about the role of the discourse argument containing a mention of entity  $j$  in sentence  $i$  (Lin et al., 2011). For example, if the first sentence is ARG1 of a comparison relation, then any entity mentions in the sentence would be labeled COMP.ARG1. This approach can also be applied to RST discourse relations (Feng et al., 2014).

**Datasets** One difficulty with evaluating metrics of discourse coherence is that human-generated texts usually meet some minimal threshold of coherence. For this reason, much of the research on measuring coherence has focused on synthetic data. A typical setting is to permute the sentences of a human-written text, and then determine whether the original sentence ordering scores higher according to the proposed coherence measure (Barzilay and Lapata, 2008). There are also small datasets of human evaluations of the coherence of machine summaries: for example, human judgments of the summaries from the participating systems in the 2003 Document Understanding Conference are available online.<sup>11</sup> Researchers from the Educational Testing Service (an organization which administers several national exams in the United States) have studied the relationship between discourse coherence and student essay quality (Burstein et al., 2003, 2010). A public dataset of essays from second-language learners, with quality annotations, has been made available by researchers at Cambridge University (Yannakoudakis et al., 2011). At the other extreme, Louis and Nenkova (2013) analyze the structure of professionally written scientific essays, finding that discourse relation transitions help to distinguish prize-winning essays from other articles in the same genre.

## Additional resources

For a manuscript-length discussion of discourse processing, see Stede (2011). Article-length surveys are offered by Webber et al. (2012) and Webber and Joshi (2012).

---

<sup>11</sup><http://homepages.inf.ed.ac.uk/mlap/coherence/>

8734 **Exercises**

- 8735    1. • Implement the smoothed cosine similarity metric from Equation 16.2, using the  
 8736        smoothing kernel  $k = [.5, .3, .15, .05]$ .  
 8737        • Download the text of a news article with at least ten paragraphs.  
 8738        • Compute and plot the smoothed similarity  $\bar{s}$  over the length of the article.  
 8739        • Identify *local minima* in  $\bar{s}$  as follows: first find all sentences  $m$  such that  $\bar{s}_m <$   
 8740         $\bar{s}_{m \pm 1}$ . Then search among these points to find the five sentences with the lowest  
 8741         $\bar{s}_m$ .  
 8742        • How often do the five local minima correspond to paragraph boundaries?  
 8743            – The fraction of local minima that are paragraph boundaries is the **precision-**  
 8744            **at- $k$** , where in this case,  $k = 5$ .  
 8745            – The fraction of paragraph boundaries which are local minima is the **recall-**  
 8746            **at- $k$** .  
 8747            – Compute precision-at- $k$  and recall-at- $k$  for  $k = 3$  and  $k = 10$ .
- 8748    2. This exercise is to be done in pairs. Each participant selects an article from to-  
 8749        day's news, and replaces all mentions of individual people with special tokens like  
 8750        PERSON1, PERSON2, and so on. The other participant should then use the rules  
 8751        of centering theory to guess each type of referring expression: full name (*Captain*  
 8752        *Ahab*), partial name (e.g., *Ahab*), nominal (e.g., *the ship's captain*), or pronoun. Check  
 8753        whether the predictions match the original article, and whether the original article  
 8754        conforms to the rules of centering theory.
- 8755    3. In § 16.3.2.1, it is noted that bottom-up parsing with compositional representations  
 8756        of each span is not guaranteed to be optimal. In this exercise, you will construct  
 8757        a minimal example proving this point. Consider a discourse with four units, with  
 8758        base representations  $\{z^{(i)}\}_{i=1}^4$ . Construct a scenario in which the parse selected by  
 8759        bottom-up parsing is not optimal, and give the precise mathematical conditions that  
 8760        must hold for this suboptimal parse to be selected. You may ignore the relation  
 8761        labels  $\ell$  for the purpose of this example.

8762

## **Part IV**

8763

# **Applications**



8764 

## Chapter 17

8765 

# Information extraction

8766 Computers offer powerful capabilities for searching and reasoning about structured records  
8767 and relational data. Some even argue that the most important limitation of artificial intel-  
8768 ligence is not inference or learning, but simply having too little knowledge (Lenat et al.,  
8769 1990). Natural language processing provides an appealing solution: automatically con-  
8770 struct a structured **knowledge base** by reading natural language text.

8771 For example, many Wikipedia pages have an “infobox” that provides structured in-  
8772 formation about an entity or event. An example is shown in Figure 17.1a: each row rep-  
8773 resents one or more properties of the entity IN THE AEROPLANE OVER THE SEA, a record  
8774 album. The set of properties is determined by a predefined **schema**, which applies to all  
8775 record albums in Wikipedia. As shown in Figure 17.1b, the values for many of these fields  
8776 are indicated directly in the first few sentences of text on the same Wikipedia page.

8777 The task of automatically constructing (or “populating”) an infobox from text is an  
8778 example of **information extraction**. Much of information extraction can be described in  
8779 terms of **entities**, **relations**, and **events**.

- 8780 • **Entities** are uniquely specified objects in the world, such as people (JEFF MANGUM),  
8781 places (ATHENS, GEORGIA), organizations (MERGE RECORDS), and times (FEBRUARY  
8782 10, 1998). Chapter 8 described the task of **named entity recognition**, which labels  
8783 tokens as parts of entity spans. Now we will see how to go further, **linking** each  
8784 entity **mention** to an element in a **knowledge base**.
- 8785 • **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	
<b>Released</b>	February 10, 1998
<b>Recorded</b>	July–September 1997
<b>Studio</b>	Pet Sounds Studio, Denver, Colorado
<b>Genre</b>	Indie rock • psychedelic folk • lo-fi
<b>Length</b>	39:55
<b>Label</b>	Merge • Domino
<b>Producer</b>	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,...>
```

8786     The set of arguments for an event type is defined by a **schema**. Events often refer to  
 8787     time-delimited occurrences: weddings, protests, purchases, terrorist attacks.

8788     Information extraction is similar to semantic role labeling (chapter 13): we may think  
 8789     of predicates as corresponding to events, and the arguments as defining slots in the event  
 8790     representation. However, the goals of information extraction are different. Rather than  
 8791     accurately parsing every sentence, information extraction systems often focus on recog-  
 8792     nizing a few key relation or event types, or on the task of identifying all properties of a  
 8793     given entity. Information extraction is often evaluated by the correctness of the resulting  
 8794     knowledge base, and not by how many sentences were accurately parsed. The goal is  
 8795     sometimes described as **macro-reading**, as opposed to **micro-reading**, in which each sen-  
 8796     tence must be analyzed correctly. Macro-reading systems are not penalized for ignoring  
 8797     difficult sentences, as long as they can recover the same information from other, easier-  
 8798     to-read sources. However, macro-reading systems must resolve apparent inconsistencies

8799 (was the album released on MERGE RECORDS or BLUE ROSE RECORDS?), requiring reasoning across the entire dataset.

8801 In addition to the basic tasks of recognizing entities, relations, and events, information  
8802 extraction systems must handle negation, and must be able to distinguish statements of  
8803 fact from hopes, fears, hunches, and hypotheticals. Finally, information extraction is often paired with the problem of **question answering**, which requires accurately parsing a  
8804 query, and then selecting or generating a textual answer. Question answering systems can  
8805 be built on knowledge bases that are extracted from large text corpora, or may attempt to  
8806 identify answers directly from the source texts.

## 8808 17.1 Entities

8809 The starting point for information extraction is to identify mentions of entities in text.  
8810 Consider the following example:

8811 (17.4) *The United States Army captured a hill overlooking Atlanta on May 14, 1864.*

8812 For this sentence, there are two goals:

- 8813 1. *Identify* the spans *United States Army*, *Atlanta*, and *May 14, 1864* as entity mentions.  
8814 (The hill is not uniquely identified, so it is not a *named* entity.) We may also want to  
8815 recognize the **named entity types**: organization, location, and date. This is **named**  
8816 **entity recognition**, and is described in chapter 8.
- 8817 2. *Link* these spans to entities in a knowledge base: U.S. ARMY, ATLANTA, and 1864-  
8818 MAY-14. This task is known as **entity linking**.

8819 The strings to be linked to entities are **mentions** — similar to the use of this term in  
8820 coreference resolution. In some formulations of the entity linking task, only named entities  
8821 are candidates for linking. This is sometimes called **named entity linking** (Ling et al.,  
8822 2015). In other formulations, such as **Wikification** (Milne and Witten, 2008), any string  
8823 can be a mention. The set of target entities often corresponds to Wikipedia pages, and  
8824 Wikipedia is the basis for more comprehensive knowledge bases such as YAGO (Suchanek  
8825 et al., 2007), DBPedia (Auer et al., 2007), and Freebase (Bollacker et al., 2008). Entity link-  
8826 ing may also be performed in more “closed” settings, where a much smaller list of targets  
8827 is provided in advance. The system must also determine if a mention does not refer to  
8828 any entity in the knowledge base, sometimes called a **NIL entity** (McNamee and Dang,  
8829 2009).

8830 Returning to (17.4), the three entity mentions may seem unambiguous. But the Wikipedia  
8831 disambiguation page for the string *Atlanta* says otherwise:<sup>1</sup> there are more than twenty

<sup>1</sup>[https://en.wikipedia.org/wiki/Atlanta\\_\(disambiguation\)](https://en.wikipedia.org/wiki/Atlanta_(disambiguation)), retrieved November 1, 2017.

8832 different towns and cities, five United States Navy vessels, a magazine, a television show,  
 8833 a band, and a singer — each prominent enough to have its own Wikipedia page. We now  
 8834 consider how to choose among these dozens of possibilities. In this chapter we will focus  
 8835 on supervised approaches. Unsupervised entity linking is closely related to the problem  
 8836 of **cross-document coreference resolution**, where the task is to identify pairs of mentions  
 8837 that corefer, across document boundaries (Bagga and Baldwin, 1998b; Singh et al., 2011).

8838 **17.1.1 Entity linking by learning to rank**

8839 Entity linking is often formulated as a **ranking** problem,

$$\hat{y} = \operatorname{argmax}_{y \in \mathcal{Y}(x)} \Psi(y, x, c), \quad [17.1]$$

8840 where  $y$  is a target entity,  $x$  is a description of the mention,  $\mathcal{Y}(x)$  is a set of candidate  
 8841 entities, and  $c$  is a description of the context — such as the other text in the document,  
 8842 or its metadata. The function  $\Psi$  is a scoring function, which could be a linear model,  
 8843  $\Psi(y, x, c) = \theta \cdot f(y, x, c)$ , or a more complex function such as a neural network. In either  
 8844 case, the scoring function can be learned by minimizing a margin-based **ranking loss**,

$$\ell(\hat{y}, y^{(i)}, x^{(i)}, c^{(i)}) = (\Psi(\hat{y}, x^{(i)}, c^{(i)}) - \Psi(y^{(i)}, x^{(i)}, c^{(i)}) + 1)_+, \quad [17.2]$$

8845 where  $y^{(i)}$  is the ground truth and  $\hat{y} \neq y^{(i)}$  is the predicted target for mention  $x^{(i)}$  in  
 8846 context  $c^{(i)}$  (Joachims, 2002; Dredze et al., 2010).

8847 **Candidate identification** For computational tractability, it is helpful to restrict the set of  
 8848 candidates,  $\mathcal{Y}(x)$ . One approach is to use a **name dictionary**, which maps from strings  
 8849 to the entities that they might mention. This mapping is many-to-many: a string such as  
 8850 *Atlanta* can refer to multiple entities, and conversely, an entity such as ATLANTA can be  
 8851 referenced by multiple strings. A name dictionary can be extracted from Wikipedia, with  
 8852 links between each Wikipedia entity page and the anchor text of all hyperlinks that point  
 8853 to the page (Bunescu and Pasca, 2006; Ratinov et al., 2011). To improve recall, the name  
 8854 dictionary can be augmented by partial and approximate matching (Dredze et al., 2010),  
 8855 but as the set of candidates grows, the risk of false positives increases. For example, the  
 8856 string *Atlanta* is a partial match to *the Atlanta Fed* (a name for the FEDERAL RESERVE BANK  
 8857 OF ATLANTA), and a noisy match (edit distance of one) from *Atalanta* (a heroine in Greek  
 8858 mythology and an Italian soccer team).

8859 **Features** Feature-based approaches to entity ranking rely on three main types of local  
 8860 information (Dredze et al., 2010):

- The similarity of the mention string to the canonical entity name, as quantified by string similarity. This feature would elevate the city ATLANTA over the basketball team ATLANTA HAWKS for the string *Atlanta*.
- The popularity of the entity, which can be measured by Wikipedia page views or PageRank in the Wikipedia link graph. This feature would elevate ATLANTA, GEORGIA over the unincorporated community of ATLANTA, OHIO.
- The entity type, as output by the named entity recognition system. This feature would elevate the city of ATLANTA over the magazine ATLANTA in contexts where the mention is tagged as a location.

In addition to these local features, the document context can also help. If *Jamaica* is mentioned in a document about the Caribbean, it is likely to refer to the island nation; in the context of New York, it is likely to refer to the neighborhood in Queens; in the context of a menu, it might refer to a hibiscus tea beverage. Such hints can be formalized by computing the similarity between the Wikipedia page describing each candidate entity and the mention context  $c^{(i)}$ , which may include the bag-of-words representing the document (Dredze et al., 2010; Hoffart et al., 2011) or a smaller window of text around the mention (Ratinov et al., 2011). For example, we can compute the cosine similarity between bag-of-words vectors for the context and entity description, typically weighted using **inverse document frequency** to emphasize rare words.<sup>2</sup>

**Neural entity linking** An alternative approach is to compute the score for each entity candidate using distributed vector representations of the entities, mentions, and context. For example, for the task of entity linking in Twitter, Yang et al. (2016) employ the bilinear scoring function,

$$\Psi(y, x, c) = v_y^\top \Theta^{(y,x)} x + v_y^\top \Theta^{(y,c)} c, \quad [17.3]$$

with  $v_y \in \mathbb{R}^{K_y}$  as the vector embedding of entity  $y$ ,  $x \in \mathbb{R}^{K_x}$  as the embedding of the mention,  $c \in \mathbb{R}^{K_c}$  as the embedding of the context, and the matrices  $\Theta^{(y,x)}$  and  $\Theta^{(y,c)}$  as parameters that score the compatibility of each entity with respect to the mention and context. Each of the vector embeddings can be learned from an end-to-end objective, or pre-trained on unlabeled data.

- Pretrained **entity embeddings** can be obtained from an existing knowledge base (Bordes et al., 2011, 2013), or by running a word embedding algorithm such as WORD2VEC

---

<sup>2</sup>The **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).

- 8891       on the text of Wikipedia, with hyperlinks substituted for the anchor text.<sup>3</sup>
- 8892     • The embedding of the mention  $x$  can be computed by averaging the embeddings  
 8893       of the words in the mention (Yang et al., 2016), or by the compositional techniques  
 8894       described in § 14.8.
- 8895     • The embedding of the context  $c$  can also be computed from the embeddings of the  
 8896       words in the context. A **denoising autoencoder** learns a function from raw text to  
 8897       dense  $K$ -dimensional vector encodings by minimizing a reconstruction loss (Vin-  
 8898       cent et al., 2010),

$$\min_{\theta_g, \theta_h} \sum_{i=1}^N \|\mathbf{x}^{(i)} - g(h(\tilde{\mathbf{x}}^{(i)}; \theta_h); \theta_g)\|^2, \quad [17.4]$$

8895       where  $\tilde{\mathbf{x}}^{(i)}$  is a noisy version of the bag-of-words counts  $\mathbf{x}^{(i)}$ , which is produced by  
 8896       randomly setting some counts to zero;  $h : \mathbb{R}^V \mapsto \mathbb{R}^K$  is an encoder with parameters  
 8897        $\theta_h$ ; and  $g : \mathbb{R}^K \mapsto \mathbb{R}^V$ , with parameters  $\theta_g$ . The encoder and decoder functions  
 8898       are typically implemented as feedforward neural networks. To apply this model to  
 8899       entity linking, each entity and context are initially represented by the encoding of  
 8900       their bag-of-words vectors,  $h(e)$  and  $g(c)$ , and these encodings are then fine-tuned  
 8901       from labeled data (He et al., 2013). The context vector  $c$  can also be obtained by  
 8902       convolution on the embeddings of words in the document (Sun et al., 2015), or by  
 8903       examining metadata such as the author’s social network (Yang et al., 2016).

8904       The remaining parameters  $\Theta^{(y,x)}$  and  $\Theta^{(y,c)}$  can be trained by backpropagation from the  
 8905       margin loss in Equation 17.2.

### 8906     17.1.2 Collective entity linking

8907       Entity linking can be more accurate when it is performed jointly across a document. To  
 8908       see why, consider the following lists:

- 8909     (17.5) California, Oregon, Washington  
 8910     (17.6) Baltimore, Washington, Philadelphia  
 8911     (17.7) Washington, Adams, Jefferson

8912       In each case, the term *Washington* refers to a different entity, and this reference is strongly  
 8913       suggested by the other entries on the list. In the last list, all three names are highly am-  
 8914       biguous — there are dozens of other *Adams* and *Jefferson* entities in Wikipedia. But a

---

<sup>3</sup>Pre-trained entity embeddings can be downloaded from <https://code.google.com/archive/p/word2vec/>.

8915 preference for coherence motivates **collectively** linking these references to the first three  
 8916 U.S. presidents.

8917 A general approach to collective entity linking is to introduce a compatibility score  
 8918  $\psi_c(\mathbf{y})$ . Collective entity linking is then performed by optimizing the global objective,

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathbb{Y}(\mathbf{x})}{\operatorname{argmax}} \Psi_c(\mathbf{y}) + \sum_{i=1}^N \Psi_\ell(y^{(i)}, \mathbf{x}^{(i)}, \mathbf{c}^{(i)}), \quad [17.5]$$

8919 where  $\mathbb{Y}(\mathbf{x})$  is the set of all possible collective entity assignments for the mentions in  $\mathbf{x}$ ,  
 8920 and  $\psi_\ell$  is the local scoring function for each entity  $i$ . The compatibility function is typically  
 8921 decomposed into a sum of pairwise scores,  $\Psi_c(\mathbf{y}) = \sum_{i=1}^N \sum_{j \neq i}^N \Psi_c(y^{(i)}, y^{(j)})$ . These scores  
 8922 can be computed in a number of different ways:

- 8923 • Wikipedia defines high-level categories for entities (e.g., *living people*, *Presidents of*  
 8924 *the United States*, *States of the United States*), and  $\Psi_c$  can reward entity pairs for the  
 8925 number of categories that they have in common (Cucerzan, 2007).
- 8926 • Compatibility can be measured by the number of incoming hyperlinks shared by  
 8927 the Wikipedia pages for the two entities (Milne and Witten, 2008).
- 8928 • In a neural architecture, the compatibility of two entities can be set equal to the inner  
 8929 product of their embeddings,  $\Psi_c(y^{(i)}, y^{(j)}) = \mathbf{v}_{y^{(i)}} \cdot \mathbf{v}_{y^{(j)}}$ .
- 8930 • A non-pairwise compatibility score can be defined using a type of latent variable  
 8931 model known as a **probabilistic topic model** (Blei et al., 2003; Blei, 2012). In this  
 8932 framework, each latent topic is a probability distribution over entities, and each  
 8933 document has a probability distribution over topics. Each entity helps to determine  
 8934 the document's distribution over topics, and in turn these topics help to resolve am-  
 8935 biguous entity mentions (Newman et al., 2006). Inference can be performed using  
 8936 the sampling techniques described in chapter 5.

8937 Unfortunately, collective entity linking is **NP-hard** even for pairwise compatibility func-  
 8938 tions, so exact optimization is almost certainly intractable. Various approximate inference  
 8939 techniques have been proposed, including **integer linear programming** (Cheng and Roth,  
 8940 2013), **Gibbs sampling** (Han and Sun, 2012), and graph-based algorithms (Hoffart et al.,  
 8941 2011; Han et al., 2011).

### 8942 17.1.3 \*Pairwise ranking loss functions

8943 The loss function defined in Equation 17.2 considers only the highest-scoring prediction  
 8944  $\hat{y}$ , but in fact, the true entity  $y^{(i)}$  should outscore *all* other entities. A loss function based on  
 8945 this idea would give a gradient against the features or representations of several entities,

**Algorithm 19** WARP approximate ranking loss

---

```

1: procedure WARP( $y^{(i)}$ ,  $\mathbf{x}^{(i)}$ )
2:    $N \leftarrow 0$ 
3:   repeat
4:     Randomly sample  $y \sim \mathcal{Y}(\mathbf{x}^{(i)})$ 
5:      $N \leftarrow N + 1$ 
6:     if  $\psi(y, \mathbf{x}^{(i)}) + 1 > \psi(y^{(i)}, \mathbf{x}^{(i)})$  then            $\triangleright$  check for margin violation
7:        $r \leftarrow \lfloor |\mathcal{Y}(\mathbf{x}^{(i)})|/N \rfloor$                           $\triangleright$  compute approximate rank
8:       return  $L_{\text{rank}}(r) \times (\psi(y, \mathbf{x}^{(i)}) + 1 - \psi(y^{(i)}, \mathbf{x}^{(i)}))$ 
9:     until  $N \geq |\mathcal{Y}(\mathbf{x}^{(i)})| - 1$                             $\triangleright$  no violation found
10:    return 0                                          $\triangleright$  return zero loss

```

---

8946 not just the top-scoring prediction. Usunier et al. (2009) define a general ranking error  
 8947 function,

$$L_{\text{rank}}(k) = \sum_{j=1}^k \alpha_j, \quad \text{with } \alpha_1 \geq \alpha_2 \geq \dots \geq 0, \quad [17.6]$$

8948 where  $k$  is equal to the number of labels ranked higher than the correct label  $y^{(i)}$ . This  
 8949 function defines a class of ranking errors: if  $\alpha_j = 1$  for all  $j$ , then the ranking error is  
 8950 equal to the rank of the correct entity; if  $\alpha_1 = 1$  and  $\alpha_{j>1} = 0$ , then the ranking error is  
 8951 one whenever the correct entity is not ranked first; if  $\alpha_j$  decreases smoothly with  $j$ , as in  
 8952  $\alpha_j = \frac{1}{j}$ , 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,

$$r(y^{(i)}, \mathbf{x}^{(i)}) \triangleq \sum_{y \in \mathcal{Y}(\mathbf{x}^{(i)}) \setminus y^{(i)}} \delta \left( 1 + \psi(y, \mathbf{x}^{(i)}) \geq \psi(y^{(i)}, \mathbf{x}^{(i)}) \right), \quad [17.7]$$

8953 where  $\delta(\cdot)$  is a delta function, and  $\mathcal{Y}(\mathbf{x}^{(i)}) \setminus y^{(i)}$  is the set of all entity candidates minus  
 8954 the true entity  $y^{(i)}$ . The margin-augmented rank is the rank of the true entity, after aug-  
 8955 menting every other candidate with a margin of one, under the current scoring function  
 8956  $\psi$ . (The context  $c$  is omitted for clarity, and can be considered part of  $x$ .)

For each instance, a hinge loss is computed from the ranking error associated with this

margin-augmented rank, and the violation of the margin constraint,

$$\ell(y^{(i)}, \mathbf{x}^{(i)}) = \frac{L_{\text{rank}}(r(y^{(i)}, \mathbf{x}^{(i)}))}{r(y^{(i)}, \mathbf{x}^{(i)})} \sum_{y \in \mathcal{Y}(\mathbf{x}) \setminus y^{(i)}} \left( \psi(y, \mathbf{x}^{(i)}) - \psi(y^{(i)}, \mathbf{x}^{(i)}) + 1 \right)_+, \quad [17.8]$$

8957 The sum in Equation 17.8 includes non-zero values for every label that is ranked at least as  
 8958 high as the true entity, after applying the margin augmentation. Dividing by the margin-  
 8959 augmented rank of the true entity thus gives the average violation.

8960 The objective in Equation 17.8 is expensive to optimize when the label space is large,  
 8961 as is usually the case for entity linking against large knowledge bases. This motivates a  
 8962 randomized approximation called **WARP** (Weston et al., 2011), shown in Algorithm 19. In  
 8963 this procedure, we sample random entities until one violates the pairwise margin con-  
 8964 straint,  $\psi(y, \mathbf{x}^{(i)}) + 1 \geq \psi(y^{(i)}, \mathbf{x}^{(i)})$ . The number of samples  $N$  required to find such  
 8965 a violation yields an approximation of the margin-augmented rank of the true entity,  
 8966  $r(y^{(i)}, \mathbf{x}^{(i)}) \approx \left\lfloor \frac{|\mathcal{Y}(\mathbf{x})|}{N} \right\rfloor$ . If a violation is found immediately,  $N = 1$ , the correct entity  
 8967 probably ranks below many others,  $r \approx |\mathcal{Y}(\mathbf{x})|$ . If many samples are required before a  
 8968 violation is found,  $N \rightarrow |\mathcal{Y}(\mathbf{x})|$ , then the correct entity is probably highly ranked,  $r \rightarrow 1$ .  
 8969 A computational advantage of WARP is that it is not necessary to find the highest-scoring  
 8970 label, which can impose a non-trivial computational cost when  $\mathcal{Y}(\mathbf{x}^{(i)})$  is large. The objec-  
 8971 tive is conceptually similar to the **negative sampling** objective in WORD2VEC (chapter 14),  
 8972 which compares the observed word against randomly sampled alternatives.

## 8973 17.2 Relations

8974 After identifying the entities that are mentioned in a text, the next step is to determine  
 8975 how they are related. Consider the following example:

8976 (17.8) George Bush traveled to France on Thursday for a summit.

8977 This sentence introduces a relation between the entities referenced by *George Bush* and  
 8978 *France*. In the Automatic Content Extraction (ACE) ontology (Linguistic Data Consortium,  
 8979 2005), the type of this relation is PHYSICAL, and the subtype is LOCATED. This relation  
 8980 would be written,

$$\text{PHYSICAL.LOCATED(GEORGE BUSH, FRANCE)}. \quad [17.9]$$

8981 Relations take exactly two arguments, and the order of the arguments matters.

8982 In the ACE datasets, relations are annotated between entity mentions, as in the exam-  
 8983 ple above. Relations can also hold between nominals, as in the following example from  
 8984 the SemEval-2010 shared task (Hendrickx et al., 2009):

CAUSE-EFFECT	<i>those cancers were caused by radiation exposures</i>
INSTRUMENT-AGENCY	<i>phone operator</i>
PRODUCT-PRODUCER	<i>a factory manufactures suits</i>
CONTENT-CONTAINER	<i>a bottle of honey was weighed</i>
ENTITY-ORIGIN	<i>letters from foreign countries</i>
ENTITY-DESTINATION	<i>the boy went to bed</i>
COMPONENT-WHOLE	<i>my apartment has a large kitchen</i>
MEMBER-COLLECTION	<i>there are many trees in the forest</i>
COMMUNICATION-TOPIC	<i>the lecture was about semantics</i>

Table 17.1: Relations and example sentences from the SemEval-2010 dataset (Hendrickx et al., 2009)

8985 (17.9) The cup contained tea from dried ginseng.

8986 This sentence describes a relation of type ENTITY-ORIGIN between *tea* and *ginseng*. Nominal  
 8987 relation extraction is closely related to **semantic role labeling** (chapter 13). The main  
 8988 difference is that relation extraction is restricted to a relatively small number of relation  
 8989 types; for example, Table 17.1 shows the ten relation types from SemEval-2010.

### 8990 17.2.1 Pattern-based relation extraction

8991 Early work on relation extraction focused on hand-crafted patterns (Hearst, 1992). For  
 8992 example, the appositive *Starbuck, a native of Nantucket* signals the relation ENTITY-ORIGIN  
 8993 between *Starbuck* and *Nantucket*. This pattern can be written as,

$$\text{PERSON , } a \text{ native of LOCATION} \Rightarrow \text{ENTITY-ORIGIN(PERSON, LOCATION)}. \quad [17.10]$$

8994 This pattern will be “triggered” whenever the literal string *, a native of* occurs between an  
 8995 entity of type PERSON and an entity of type LOCATION. Such patterns can be generalized  
 8996 beyond literal matches using techniques such as lemmatization, which would enable the  
 8997 words (*buy, buys, buying*) to trigger the same patterns (see § 4.3.1.2). A more aggressive  
 8998 strategy would be to group all words in a WordNet synset (§ 4.2), so that, e.g., *buy* and  
 8999 *purchase* trigger the same patterns.

9000 Relation extraction patterns can be implemented in finite-state automata (§ 9.1). If the  
 9001 named entity recognizer is also a finite-state machine, then the systems can be combined  
 9002 by finite-state transduction (Hobbs et al., 1997). This makes it possible to propagate uncer-  
 9003 tainty through the finite-state cascade, and disambiguate from higher-level context. For  
 9004 example, suppose the entity recognizer cannot decide whether *Starbuck* refers to either a  
 9005 PERSON or a LOCATION; in the composed transducer, the relation extractor would be free  
 9006 to select the PERSON annotation when it appears in the context of an appropriate pattern.

9007 **17.2.2 Relation extraction as a classification task**

9008 Relation extraction can be formulated as a classification problem,

$$\hat{r}_{(i,j),(m,n)} = \operatorname{argmax}_{r \in \mathcal{R}} \Psi(r, (i, j), (m, n), \mathbf{w}), \quad [17.11]$$

9009 where  $r \in \mathcal{R}$  is a relation type (possibly NIL),  $\mathbf{w}_{i+1:j}$  is the span of the first argument, and  
 9010  $\mathbf{w}_{m+1:n}$  is the span of the second argument. The argument  $\mathbf{w}_{m+1:n}$  may appear before  
 9011 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  
 9012 argument of the relation. We now consider three alternatives for computing the scoring  
 9013 function.

9014 **17.2.2.1 Feature-based classification**

9015 In a feature-based classifier, the scoring function is defined as,

$$\Psi(r, (i, j), (m, n), \mathbf{w}) = \boldsymbol{\theta} \cdot \mathbf{f}(r, (i, j), (m, n), \mathbf{w}), \quad [17.12]$$

9016 with  $\boldsymbol{\theta}$  representing a vector of weights, and  $\mathbf{f}(\cdot)$  a vector of features. The pattern-based  
 9017 methods described in § 17.2.1 suggest several features:

- 9018 • Local features of  $\mathbf{w}_{i+1:j}$  and  $\mathbf{w}_{m+1:n}$ , including: the strings themselves; whether they  
 9019 are recognized as entities, and if so, which type; whether the strings are present in a  
 9020 **gazetteer** of entity names; each string's syntactic **head** (§ 9.2.2).
- 9021 • Features of the span between the two arguments,  $\mathbf{w}_{j+1:m}$  or  $\mathbf{w}_{n+1:i}$  (depending on  
 9022 which argument appears first): the length of the span; the specific words that appear  
 9023 in the span, either as a literal sequence or a bag-of-words; the wordnet synsets (§ 4.2)  
 9024 that appear in the span between the arguments.
- 9025 • Features of the syntactic relationship between the two arguments, typically the **de-**  
 9026 **pendency path** between the arguments (§ 13.2.1). Example dependency paths are  
 9027 shown in Table 17.2.

9028 **17.2.2.2 Kernels**

9029 Suppose that the first line of Table 17.2 is a labeled example, and the remaining lines are  
 9030 instances to be classified. A feature-based approach would have to decompose the depen-  
 9031 dency paths into features that capture individual edges, with or without their labels, and  
 9032 then learn weights for each of these features: for example, the second line contains identi-  
 9033 cal dependencies, but different arguments; the third line contains a different inflection of  
 9034 the word *travel*; the fourth and fifth lines each contain an additional edge on the depen-  
 9035 dency path; and the sixth example uses an entirely different path. Rather than attempting  
 9036 to create local features that capture all of the ways in which these dependencies paths

1. <i>George Bush traveled to France</i>	<i>George Bush</i> $\leftarrow$ traveled $\rightarrow$ France NSUBJ OBL
2. <i>Ahab traveled to Nantucket</i>	<i>Ahab</i> $\leftarrow$ traveled $\rightarrow$ Nantucket NSUBJ OBL
3. <i>George Bush will travel to France</i>	<i>George Bush</i> $\leftarrow$ travel $\rightarrow$ France NSUBJ OBL
4. <i>George Bush wants to travel to France</i>	<i>George Bush</i> $\leftarrow$ wants $\rightarrow$ travel $\rightarrow$ France NSUBJ XCOMP OBL
5. <i>Ahab traveled to a city in France</i>	<i>Ahab</i> $\leftarrow$ traveled $\rightarrow$ city $\rightarrow$ France NSUBJ OBL NMOD
6. <i>We await Ahab's visit to France</i>	<i>Ahab</i> $\leftarrow$ visit $\rightarrow$ France NMOD:POSS NMOD

Table 17.2: Candidates instances for the PHYSICAL.LOCATED relation, and their dependency paths

9037 are similar and different, we can instead define a similarity function  $\kappa$ , which computes a  
 9038 score for any pair of instances,  $\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}_+$ . The score for any pair of instances  $(i, j)$   
 9039 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  
 9040 function  $\kappa$  obeys a few key properties it is a valid **kernel function**.<sup>4</sup>

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,

$$\hat{y} = \text{Sign}(b + \sum_{i=1}^N y^{(i)} \alpha^{(i)} \kappa(\mathbf{x}^{(i)}, \mathbf{x})) \quad [17.13]$$

9041 where  $b$  and  $\{\alpha^{(i)}\}_{i=1}^N$  are parameters that must be learned from the training set, under  
 9042 the constraint  $\forall_i, \alpha^{(i)} \geq 0$ . Intuitively, each  $\alpha_i$  specifies the importance of the instance  $\mathbf{x}^{(i)}$   
 9043 towards the classification rule. Kernel-based classification can be viewed as a weighted  
 9044 form of the **nearest-neighbor** classifier (Hastie et al., 2009), in which test instances are  
 9045 assigned the most common label among their near neighbors in the training set. This  
 9046 results in a non-linear classification boundary. The parameters are typically learned from  
 9047 a margin-based objective (see § 2.3), leading to the **kernel support vector machine**. To  
 9048 generalize to multi-class classification, we can train separate binary classifiers for each  
 9049 label (sometimes called **one-versus-all**), or train binary classifiers for each pair of possible  
 9050 labels (**one-versus-one**).

9051 Dependency kernels are particularly effective for relation extraction, due to their ability  
 9052 to capture syntactic properties of the path between the two candidate arguments. One  
 9053 class of dependency tree kernels is defined recursively, with the score for a pair of trees

<sup>4</sup>The **Gram matrix**  $\mathbf{K}$  arises from computing the kernel function between all pairs in a set of instances. For a valid kernel, the Gram matrix must be symmetric ( $\mathbf{K} = \mathbf{K}^\top$ ) and positive semi-definite ( $\forall \mathbf{a}, \mathbf{a}^\top \mathbf{K} \mathbf{a} \geq 0$ ). For more on kernel-based classification, see chapter 14 of Murphy (2012).

equal to the similarity of the root nodes and the sum of similarities of matched pairs of child subtrees (Zelenko et al., 2003; Culotta and Sorensen, 2004). Alternatively, Bunescu and Mooney (2005) define a kernel function over sequences of unlabeled dependency edges, in which the score is computed as a product of scores for each pair of words in the sequence: identical words receive a high score, words that share a synset or part-of-speech receive a small non-zero score (e.g., *travel* / *visit*), and unrelated words receive a score of zero.

### 17.2.2.3 Neural relation extraction

**Convolutional neural networks** were an early neural architecture for relation extraction (Zeng et al., 2014; dos Santos et al., 2015). For the sentence  $(w_1, w_2, \dots, w_M)$ , obtain a matrix of word embeddings  $\mathbf{X}$ , where  $x_m \in \mathbb{R}^K$  is the embedding of  $w_m$ . Now, suppose the candidate arguments appear at positions  $a_1$  and  $a_2$ ; then for each word in the sentence, its position with respect to each argument is  $m - a_1$  and  $m - a_2$ . (Following Zeng et al. (2014), this is a restricted version of the relation extraction task in which the arguments are single tokens.) To capture any information conveyed by these positions, the word embeddings are concatenated with embeddings of the positional offsets,  $x_{m-a_1}^{(p)}$  and  $x_{m-a_2}^{(p)}$ . The complete base representation of the sentence is,

$$\mathbf{X}(a_1, a_2) = \begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_M \\ \mathbf{x}_{1-a_1}^{(p)} & \mathbf{x}_{2-a_1}^{(p)} & \cdots & \mathbf{x}_{M-a_1}^{(p)} \\ \mathbf{x}_{1-a_2}^{(p)} & \mathbf{x}_{2-a_2}^{(p)} & \cdots & \mathbf{x}_{M-a_2}^{(p)} \end{pmatrix}, \quad [17.14]$$

where each column is a vertical concatenation of a word embedding, represented by the column vector  $\mathbf{x}_m$ , and two positional embeddings, specifying the position with respect to  $a_1$  and  $a_2$ . The matrix  $\mathbf{X}(a_1, a_2)$  is then taken as input to a convolutional layer (see § 3.4), and max-pooling is applied to obtain a vector. The final scoring function is then,

$$\Psi(r, i, j, \mathbf{X}) = \theta_r \cdot \text{MaxPool}(\text{ConvNet}(\mathbf{X}(i, j); \phi)), \quad [17.15]$$

where  $\phi$  defines the parameters of the convolutional operator, and the  $\theta_r$  defines a set of weights for relation  $r$ . The model can be trained using a margin objective,

$$\hat{r} = \underset{r}{\operatorname{argmax}} \Psi(r, i, j, \mathbf{X}) \quad [17.16]$$

$$\ell = (1 + \psi(\hat{r}, i, j, \mathbf{X}) - \psi(r, i, j, \mathbf{X}))_+. \quad [17.17]$$

**Recurrent neural networks** have also been applied to relation extraction, using a network such as an bidirectional LSTM to encode the words or dependency path between the two arguments. Xu et al. (2015) segment each dependency path into left and right subpaths: the path  $George \xleftarrow{\text{NSUBJ}} Bush \xrightarrow{\text{XCOMP}} wants \xrightarrow{\text{OBL}} travel \rightarrow France$  is segmented into the subpaths,

9080 (17.10) *George Bush*  $\xleftarrow[\text{NSUBJ}]{} \text{wants}$

9081 (17.11)  $\text{wants} \xrightarrow[\text{XCOMP}]{} \text{travel} \xrightarrow[\text{OBL}]{} \text{France.}$

Xu et al. (2015) then run recurrent networks from the arguments to the root word (in this case, *wants*), obtaining the final representation by max pooling across all the recurrent states along each path. This process can be applied across separate “channels”, in which the inputs consist of embeddings for the words, parts-of-speech, dependency relations, and WordNet hypernyms. To define the model formally, let  $s(m)$  define the successor of word  $m$  in either the left or right subpath (in a dependency path, each word can have a successor in at most one subpath). Let  $\mathbf{x}_m^{(c)}$  indicate the embedding of word (or relation)  $m$  in channel  $c$ , and let  $\overleftarrow{\mathbf{h}}_m^{(c)}$  and  $\overrightarrow{\mathbf{h}}_m^{(c)}$  indicate the associated recurrent states in the left and right subtrees respectively. Then the complete model is specified as follows,

$$\mathbf{h}_{s(m)}^{(c)} = \text{RNN}(\mathbf{x}_{s(m)}^{(c)}, \mathbf{h}_m^{(c)}) \quad [17.18]$$

$$\mathbf{z}^{(c)} = \text{MaxPool} \left( \overleftarrow{\mathbf{h}}_i^{(c)}, \overleftarrow{\mathbf{h}}_{s(i)}^{(c)}, \dots, \overleftarrow{\mathbf{h}}_{\text{root}}^{(c)}, \overrightarrow{\mathbf{h}}_j^{(c)}, \overrightarrow{\mathbf{h}}_{s(j)}^{(c)}, \dots, \overrightarrow{\mathbf{h}}_{\text{root}}^{(c)} \right) \quad [17.19]$$

$$\Psi(r, i, j) = \theta \cdot \left[ \mathbf{z}^{(\text{word})}; \mathbf{z}^{(\text{POS})}; \mathbf{z}^{(\text{dependency})}; \mathbf{z}^{(\text{hypernym})} \right]. \quad [17.20]$$

9082 Note that  $\mathbf{z}$  is computed by applying max-pooling to the *matrix* of horizontally concatenated vectors  $\mathbf{h}$ , while  $\Psi$  is computed from the *vector* of vertically concatenated vectors  $\mathbf{z}$ . Xu et al. (2015) pass the score  $\Psi$  through a **softmax** layer to obtain a probability  $p(r | i, j, w)$ , and train the model by regularized **cross-entropy**. Miwa and Bansal (2016) show that a related model can solve the more challenging “end-to-end” relation extraction task, in which the model must simultaneously detect entities and then extract their relations.

### 9089 17.2.3 Knowledge base population

9090 In many applications, what matters is not what fraction of sentences are analyzed correctly, but how much accurate knowledge can be extracted. **Knowledge base population** (KBP) refers to the task of filling in Wikipedia-style infoboxes, as shown in Figure 17.1a. 9092 Knowledge base population can be decomposed into two subtasks: **entity linking** (described in § 17.1), and **slot filling** (Ji and Grishman, 2011). Slot filling has two key differences from the formulation of relation extraction presented above: the relations hold 9094 between entities rather than spans of text, and the performance is evaluated at the *type level* (on entity pairs), rather than on the *token level* (on individual sentences).

9098 From a practical standpoint, there are three other important differences between slot 9099 filling and per-sentence relation extraction.

- KBP tasks are often formulated from the perspective of identifying attributes of a few “query” entities. As a result, these systems often start with an **information retrieval** phase, in which relevant passages of text are obtained by search.
- For many entity pairs, there will be multiple passages of text that provide evidence. Slot filling systems must aggregate this evidence to predict a single relation type (or set of relations).
- Labeled data is usually available in the form of pairs of related entities, rather than annotated passages of text. Training from such type-level annotations is a challenge: two entities may be linked by several relations, or they may appear together in a passage of text that nonetheless does not describe their relation to each other.

Information retrieval is beyond the scope of this text (see Manning et al., 2008). The remainder of this section describes approaches to information fusion and learning from type-level annotations.

### 17.2.3.1 Information fusion

In knowledge base population, there will often be multiple pieces of evidence for (and sometimes against) a single relation. For example, a search for the entity MAYNARD JACKSON, JR. may return several passages that reference the entity ATLANTA:<sup>5</sup>

- (17.12) Elected mayor of Atlanta in 1973, **Maynard Jackson** was the first African American to serve as mayor of a major southern city.
- (17.13) **Atlanta**’s airport will be renamed to honor **Maynard Jackson**, the city’s first Black mayor .
- (17.14) Born in Dallas, Texas in 1938, **Maynard Holbrook Jackson, Jr.** moved to Atlanta when he was 8.
- (17.15) **Maynard Jackson** has gone from one of the worst high schools in **Atlanta** to one of the best.

The first and second examples provide evidence for the relation **MAYOR** holding between the entities **ATLANTA** and **MAYNARD JACKSON, JR.**. The third example provides evidence for a different relation between these same entities, **LIVED-IN**. The fourth example poses an entity linking problem, referring to **MAYNARD JACKSON HIGH SCHOOL**. Knowledge base population requires aggregating this sort of textual evidence, and predicting the relations that are most likely to hold.

---

<sup>5</sup>First 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](http://www.todayingeorgiahistory.org/content/maynard-jackson-elected)

9131 One approach is to run a single-document relation extraction system (using the tech-  
 9132 niques described in § 17.2.2), and then aggregate the results (Li et al., 2011). Relations  
 9133 that are detected with high confidence in multiple documents are more likely to be valid,  
 9134 motivating the heuristic,

$$\psi(r, e_1, e_2) = \sum_{i=1}^N (\text{p}(r(e_1, e_2) | \mathbf{w}^{(i)}))^\alpha, \quad [17.21]$$

9135 where  $\text{p}(r(e_1, e_2) | \mathbf{w}^{(i)})$  is the probability of relation  $r$  between entities  $e_1$  and  $e_2$  condi-  
 9136 tioned on the text  $\mathbf{w}^{(i)}$ , and  $\alpha \gg 1$  is a tunable hyperparameter. Using this heuristic, it is  
 9137 possible to rank all candidate relations, and trace out a **precision-recall curve** as more rel-  
 9138 ations are extracted.<sup>6</sup> Alternatively, features can be aggregated across multiple passages  
 9139 of text, feeding a single type-level relation extraction system (Wolfe et al., 2017).

9140 Precision can be improved by introducing constraints across multiple relations. For  
 9141 example, if we are certain of the relation  $\text{PARENT}(e_1, e_2)$ , then it cannot also be the case  
 9142 that  $\text{PARENT}(e_2, e_1)$ . Integer linear programming makes it possible to incorporate such  
 9143 constraints into a global optimization (Li et al., 2011). Other pairs of relations have pos-  
 9144 itive correlations, such  $\text{MAYOR}(e_1, e_2)$  and  $\text{LIVED-IN}(e_1, e_2)$ . Compatibility across relation  
 9145 types can be incorporated into probabilistic graphical models (e.g., Riedel et al., 2010).

### 9146 17.2.3.2 Distant supervision

9147 Relation extraction is “annotation hungry,” because each relation requires its own la-  
 9148 beled data. Rather than relying on annotations of individual documents, it would be  
 9149 preferable to use existing knowledge resources — such as the many facts that are al-  
 9150 ready captured in knowledge bases like DBpedia. However such annotations raise the  
 9151 inverse of the information fusion problem considered above: the existence of the relation  
 9152  $\text{MAYOR}(\text{MAYNARD JACKSON JR., ATLANTA})$  provides only **distant supervision** for the  
 9153 example texts in which this entity pair is mentioned.

9154 One approach is to treat the entity pair as the instance, rather than the text itself (Mintz  
 9155 et al., 2009). Features are then aggregated across all sentences in which both entities are  
 9156 mentioned, and labels correspond to the relation (if any) between the entities in a knowl-  
 9157 edge base, such as FreeBase. Negative instances are constructed from entity pairs that are  
 9158 not related in the knowledge base. In some cases, two entities are related, but the knowl-  
 9159 edge base is missing the relation; however, because the number of possible entity pairs is  
 9160 huge, these missing relations are presumed to be relatively rare. This approach is shown  
 9161 in Figure 17.2.

---

<sup>6</sup>The 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.

9162        In **multiple instance learning**, labels are assigned to *sets* of instances, of which only  
 9163        an unknown subset are actually relevant (Dietterich et al., 1997; Maron and Lozano-Pérez,  
 9164        1998). This formalizes the framework of distant supervision: the relation  $\text{REL}(A, B)$  acts  
 9165        as a label for the entire set of sentences mentioning entities A and B, even when only a  
 9166        subset of these sentences actually describes the relation. One approach to multi-instance  
 9167        learning is to introduce a binary **latent variable** for each sentence, indicating whether the  
 9168        sentence expresses the labeled relation (Riedel et al., 2010). A variety of inference tech-  
 9169        niques have been employed for this probabilistic model of relation extraction: Surdeanu  
 9170        et al. (2012) use expectation maximization, Riedel et al. (2010) use sampling, and Hoff-  
 9171        mann et al. (2011) use a custom graph-based algorithm. Expectation maximization and  
 9172        sampling are surveyed in chapter 5, and are covered in more detail by Murphy (2012);  
 9173        graph-based methods are surveyed by Mihalcea and Radev (2011).

Task	Relation ontology	Supervision
PropBank semantic role labeling	VerbNet	sentence
FrameNet semantic role labeling	FrameNet	sentence
Relation extraction	ACE, TAC, SemEval, etc	sentence
Slot filling	ACE, TAC, SemEval, etc	relation
Open Information Extraction	open	seed relations or patterns

Table 17.3: Various relation extraction tasks and their properties. VerbNet and FrameNet are described in chapter 13. ACE (Linguistic Data Consortium, 2005), TAC (McNamee and Dang, 2009), and SemEval (Hendrickx et al., 2009) refer to shared tasks, each of which involves an ontology of relation types.

#### 9174 17.2.4 Open information extraction

9175 In classical relation extraction, the set of relations is defined in advance, using a **schema**.  
 9176 The relation for any pair of entities can then be predicted using multi-class classification.  
 9177 In **open information extraction** (OpenIE), a relation can be any triple of text. The example  
 9178 sentence (17.12) instantiates several “relations” of this sort:

- 9179 • (*mayor of, Maynard Jackson, Atlanta*),
- 9180 • (*elected, Maynard Jackson, mayor of Atlanta*),
- 9181 • (*elected in, Maynard Jackson, 1973*),

9182 and so on. Extracting such tuples can be viewed as a lightweight version of **semantic role**  
 9183 **labeling** (chapter 13), with only two argument types: first slot and second slot. The task is  
 9184 generally evaluated on the relation level, rather than on the level of sentences: precision is  
 9185 measured by the number of extracted relations that are accurate, and recall is measured by  
 9186 the number of true relations that were successfully extracted. OpenIE systems are trained  
 9187 from distant supervision or bootstrapping, rather than from labeled sentences.

9188 An early example is the TextRunner system (Banko et al., 2007), which identifies re-  
 9189 lations with a set of handcrafted syntactic rules. The examples that are acquired from the  
 9190 handcrafted rules are then used to train a classification model that uses part-of-speech pat-  
 9191 terns as features. Finally, the relations that are extracted by the classifier are aggregated,  
 9192 removing redundant relations and computing the number of times that each relation is  
 9193 mentioned in the corpus. TextRunner was the first in a series of systems that performed  
 9194 increasingly accurate open relation extraction by incorporating more precise linguistic fea-  
 9195 tures (Etzioni et al., 2011), distant supervision from Wikipedia infoboxes (Wu and Weld,  
 9196 2010), and better learning algorithms (Zhu et al., 2009).

**9197 17.3 Events**

9198 Relations link pairs of entities, but many real-world situations involve more than two en-  
9199 tities. Consider again the example sentence (17.12), which describes the **event** of an elec-  
9200 tion, with four properties: the office (MAYOR), the district (ATLANTA), the date (1973), and  
9201 the person elected (MAYNARD JACKSON, JR.). In **event detection**, a schema is provided  
9202 for each event type (e.g., an election, a terrorist attack, or a chemical reaction), indicating  
9203 all the possible properties of the event. The system is then required to fill in as many of  
9204 these properties as possible (Doddington et al., 2004).

9205 Event detection systems generally involve a retrieval component (finding relevant  
9206 documents and passages of text) and an extraction component (determining the proper-  
9207 ties of the event based on the retrieved texts). Early approaches focused on finite-state pat-  
9208 terns for identify event properties (Hobbs et al., 1997); such patterns can be automatically  
9209 induced by searching for patterns that are especially likely to appear in documents that  
9210 match the event query (Riloff, 1996). Contemporary approaches employ techniques that  
9211 are similar to FrameNet semantic role labeling (§ 13.2), such as structured prediction over  
9212 local and global features (Li et al., 2013) and bidirectional recurrent neural networks (Feng  
9213 et al., 2016). These methods detect whether an event is described in a sentence, and if so,  
9214 what are its properties.

9215 **Event coreference** Because multiple sentences may describe unique properties of a sin-  
9216 gle event, **event coreference** is required to link event mentions across a single passage  
9217 of text, or between passages (Humphreys et al., 1997). Bejan and Harabagiu (2014) de-  
9218 fine event coreference as the task of identifying event mentions that share the same event  
9219 participants (i.e., the slot-filling entities) and the same event properties (e.g., the time and  
9220 location), within or across documents. Event coreference resolution can be performed us-  
9221 ing supervised learning techniques in a similar way to entity coreference, as described  
9222 in chapter 15: move left-to-right through the document, and use a classifier to decide  
9223 whether to link each event reference to an existing cluster of coreferent events, or to cre-  
9224 ate a new cluster (Ahn, 2006). Each clustering decision is based on the compatibility of  
9225 features describing the participants and properties of the event. Due to the difficulty of  
9226 annotating large amounts of data for entity coreference, unsupervised approaches are es-  
9227 pecially desirable (Chen and Ji, 2009; Bejan and Harabagiu, 2014).

9228 **Relations between events** Just as entities are related to other entities, events may be  
9229 related to other events: for example, the event of winning an election both *precedes* and  
9230 *causes* the event of serving as mayor; moving to Atlanta *precedes* and *enables* the event of  
9231 becoming mayor of Atlanta; moving from Dallas to Atlanta *prevents* the event of later be-  
9232 coming mayor of Dallas. As these examples show, events may be related both temporally  
9233 and causally. The **TimeML** annotation scheme specifies a set of six temporal relations

	Positive (+)	Negative (-)	Underspecified (u)
Certain (CT)	Fact: CT+	Counterfact: CT-	Certain, but unknown: CTU
Probable (PR)	Probable: PR+	Not probable: PR-	(NA)
Possible (PS)	Possible: PS+	Not possible: PS-	(NA)
Underspecified (U)	(NA)	(NA)	Unknown or uncommitted: UU

Table 17.4: Table of factuality values from the FactBank corpus (Saurí and Pustejovsky, 2009). The entry (NA) indicates that this combination is not annotated.

9234 between events (Pustejovsky et al., 2005), derived in part from **interval algebra** (Allen,  
9235 1984). The TimeBank corpus provides TimeML annotations for 186 documents (Pustejovsky  
9236 et al., 2003). Methods for detecting these temporal relations combine supervised  
9237 machine learning with temporal constraints, such as transitivity (e.g. Mani et al., 2006;  
9238 Chambers and Jurafsky, 2008).

9239 More recent annotation schemes and datasets combine temporal and causal relations (Mirza  
9240 et al., 2014; Dunietz et al., 2017): for example, the CaTeRS dataset includes annotations of  
9241 320 five-sentence short stories (Mostafazadeh et al., 2016). Abstracting still further, **processes**  
9242 are networks of causal relations between multiple events. A small dataset of biological  
9243 processes is annotated in the ProcessBank dataset (Berant et al., 2014), with the  
9244 goal of supporting automatic question answering on scientific textbooks.

## 9245 17.4 Hedges, denials, and hypotheticals

9246 The methods described thus far apply to **propositions** about the way things are in the  
9247 real world. But natural language can also describe events and relations that are likely or  
9248 unlikely, possible or impossible, desired or feared. The following examples hint at the  
9249 scope of the problem (Prabhakaran et al., 2010):

- 9250 (17.16) GM will lay off workers.
- 9251 (17.17) A spokesman for GM said GM will lay off workers.
- 9252 (17.18) GM may lay off workers.
- 9253 (17.19) The politician claimed that GM will lay off workers.
- 9254 (17.20) Some wish GM would lay off workers.
- 9255 (17.21) Will GM lay off workers?
- 9256 (17.22) Many wonder whether GM will lay off workers.

9257 Accurate information extraction requires handling these **extra-propositional** aspects  
9258 of meaning, which are sometimes summarized under the terms **modality** and **negation**.<sup>7</sup>  
9259 Modality refers to expressions of the speaker's attitude towards her own statements, in-  
9260 cluding "degree of certainty, reliability, subjectivity, sources of information, and perspec-  
9261 tive" (Morante and Sporleder, 2012). Various systematizations of modality have been  
9262 proposed (e.g., Palmer, 2001), including categories such as future, interrogative, imper-  
9263 ative, conditional, and subjective. Information extraction is particularly concerned with  
9264 negation and certainty. For example, Saurí and Pustejovsky (2009) link negation with  
9265 a modal calculus of certainty, likelihood, and possibility, creating the two-dimensional  
9266 schema shown in Table 17.4. This is the basis for the FactBank corpus, with annotations  
9267 of the **factuality** of all sentences in 208 documents of news text.

9268 A related concept is **hedging**, in which speakers limit their commitment to a proposi-  
9269 tion (Lakoff, 1973):

- 9270 (17.23) These results **suggest** that expression of c-jun, jun B and jun D genes **might** be in-  
9271 volved in terminal granulocyte differentiation... (Morante and Daelemans, 2009)  
9272 (17.24) A whale is **technically** a mammal (Lakoff, 1973)

9273 In the first example, the hedges *suggest* and *might* communicate uncertainty; in the second  
9274 example, there is no uncertainty, but the hedge *technically* indicates that the evidence for  
9275 the proposition will not fully meet the reader's expectations. Hedging has been studied  
9276 extensively in scientific texts (Medlock and Briscoe, 2007; Morante and Daelemans, 2009),  
9277 where the goal of large-scale extraction of scientific facts is obstructed by hedges and spec-  
9278 ulation. Still another related aspect of modality is **evidentiality**, in which speakers mark  
9279 the source of their information. In many languages, it is obligatory to mark evidentiality  
9280 through affixes or particles (Aikhenvald, 2004); while evidentiality is not grammaticalized  
9281 in English, authors are expected to express this information in contexts such as journal-  
9282 ism (Kovach and Rosenstiel, 2014) and Wikipedia.<sup>8</sup>

9283 Methods for handling negation and modality generally include two phases:

- 9284 1. detecting negated or uncertain events;  
9285 2. identifying the scope and focus of the negation or modal operator.

---

<sup>7</sup>The 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.

<sup>8</sup><https://en.wikipedia.org/wiki/Wikipedia:Verifiability>

9286 A considerable body of work on negation has employed rule-based techniques such as  
 9287 regular expressions (Chapman et al., 2001) to detect negated events. Such techniques  
 9288 match lexical cues (e.g., *Norwood was not elected Mayor*), while avoiding “double nega-  
 9289 tives” (e.g., *surely all this is not without meaning*). More recent approaches employ classi-  
 9290 fiers over lexical and syntactic features (Uzuner et al., 2009) and sequence labeling (Prab-  
 9291 hakaran et al., 2010).

9292 The tasks of scope and focus resolution are more fine grained, as shown in the example  
 9293 from Morante and Sporleder (2012):

- 9294 (17.25) [ After his habit he said ] **nothing**, and after mine I asked no questions.  
 9295       After his habit he said nothing, and [ after mine I asked ] **no** [ questions ].

9296 In this sentence, there are two negation cues (*nothing* and *no*). Each negates an event,  
 9297 indicated by the underlined verbs *said* and *asked* (this is the focus of negation), and each  
 9298 occurs within a scope: *after his habit he said* and *after mine I asked* \_\_\_\_ *questions*. These tasks  
 9299 are typically formalized as sequence labeling problems, with each word token labeled  
 9300 as beginning, inside, or outside of a cue, focus, or scope span (see § 8.3). Conventional  
 9301 sequence labeling approaches can then be applied, using surface features as well as syn-  
 9302 tax (Velldal et al., 2012) and semantic analysis (Packard et al., 2014). Labeled datasets  
 9303 include the BioScope corpus of biomedical texts (Vincze et al., 2008) and a shared task  
 9304 dataset of detective stories by Arthur Conan Doyle (Morante and Blanco, 2012).

## 9305 17.5 Question answering and machine reading

9306 The victory of the Watson question-answering system against three top human players on  
 9307 the game show *Jeopardy!* was a landmark moment for natural language processing (Fer-  
 9308 rucci et al., 2010). Game show questions are usually answered by **factoids**: entity names  
 9309 and short phrases.<sup>9</sup> The task of factoid question answering is therefore closely related to  
 9310 information extraction, with the additional problem of accurately parsing the question.

### 9311 17.5.1 Formal semantics

9312 Semantic parsing is an effective method for question-answering in restricted domains  
 9313 such as questions about geography and airline reservations (Zettlemoyer and Collins,  
 9314 2005), and has also been applied in “open-domain” settings such as question answering  
 9315 on Freebase (Berant et al., 2013) and biomedical research abstracts (Poon and Domingos,  
 9316 2009). One approach is to convert the question into a lambda calculus expression that

---

<sup>9</sup>The 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).

9317 returns a boolean value: for example, the question *who is the mayor of the capital of Georgia?*  
 9318 would be converted to,

$$\lambda x. \exists y \text{ CAPITAL(GEORGIA, } y) \wedge \text{MAYOR}(y, x). \quad [17.22]$$

9319 This lambda expression can then be used to query an existing knowledge base, returning  
 9320 “true” for all entities that satisfy it.

### 9321 17.5.2 Machine reading

9322 Recent work has focused on answering questions about specific textual passages, similar  
 9323 to the reading comprehension examinations for young students (Hirschman et al., 1999).  
 9324 This task has come to be known as **machine reading**.

#### 9325 17.5.2.1 Datasets

9326 The machine reading problem can be formulated in a number of different ways. The most  
 9327 important distinction is what form the answer should take.

- 9328 • **Multiple-choice question answering**, as in the MCTest dataset of stories (Richardson et al., 2013) and the New York Regents Science Exams (Clark, 2015). In MCTest,  
 9329 the answer is deducible from the text alone, while in the science exams, the system  
 9330 must make inferences using an existing model of the underlying scientific phenomena.  
 9331 Here is an example from MCTest:

9333 (17.26) James the turtle was always getting into trouble. Sometimes he'd reach into  
 9334 the freezer and empty out all the food ...

9335 Q: What is the name of the trouble making turtle?  
 9336 (a) Fries  
 9337 (b) Pudding  
 9338 (c) James  
 9339 (d) Jane

- 9340 • **Cloze-style “fill in the blank”** questions, as in the CNN/Daily Mail comprehension  
 9341 task (Hermann et al., 2015), the Children’s Book Test (Hill et al., 2016), and the Who-  
 9342 did-What dataset (Onishi et al., 2016). In these tasks, the system must guess which  
 9343 word or entity completes a sentence, based on reading a passage of text. Here is an  
 9344 example from Who-did-What:

9345 (17.27) Q: Tottenham manager Juande Ramos has hinted he will allow \_\_\_\_ to leave  
 9346 if the Bulgaria striker makes it clear he is unhappy. (Onishi et al., 2016)

9347     The query sentence may be selected either from the story itself, or from an external  
 9348     summary. In either case, datasets can be created automatically by processing large  
 9349     quantities existing documents. An additional constraint is that that missing element  
 9350     from the cloze must appear in the main passage of text: for example, in Who-did-  
 9351     What, the candidates include all entities mentioned in the main passage. In the  
 9352     CNN/Daily Mail dataset, each entity name is replaced by a unique identifier, e.g.,  
 9353     ENTITY37. This ensures that correct answers can only be obtained by accurately  
 9354     reading the text, and not from external knowledge about the entities.

- 9355     • **Extractive** question answering, in which the answer is drawn from the original text.  
 9356     In WikiQA, answers are sentences (Yang et al., 2015). In the Stanford Question An-  
 9357     swering Dataset (SQuAD), answers are words or short phrases (Rajpurkar et al.,  
 9358     2016):

9359       (17.28) In metereology, precipitation is any product of the condensation of atmo-  
 9360        spheric water vapor that falls under gravity.  
 9361        Q: What causes precipitation to fall? A: gravity

9362       In both WikiQA and SQuAD, the original texts are Wikipedia articles, and the ques-  
 9363       tions are generated by crowdworkers.

9364     **17.5.2.2 Methods**

9365       A baseline method is to search the text for sentences or short passages that overlap with  
 9366       both the query and the candidate answer (Richardson et al., 2013). In example (17.26), this  
 9367       baseline would select the correct answer, since *James* appears in a sentence that includes  
 9368       the query terms *trouble* and *turtle*.

This baseline can be implemented as a neural architecture, using an **attention mechanism** (see § 18.3.1), which scores the similarity of the query to each part of the source text (Chen et al., 2016). The first step is to encode the passage  $w^{(p)}$  and the query  $w^{(q)}$ , using two bidirectional LSTMs (§ 7.6).

$$\mathbf{h}^{(q)} = \text{BiLSTM}(\mathbf{w}^{(q)}; \Theta^{(q)}) \quad [17.23]$$

$$\mathbf{h}^{(p)} = \text{BiLSTM}(\mathbf{w}^{(p)}; \Theta^{(p)}). \quad [17.24]$$

The query is represented by vertically concatenating the final states of the left-to-right and right-to-left passes:

$$\mathbf{u} = [\overrightarrow{\mathbf{h}}^{(q)}_{M_q}; \overleftarrow{\mathbf{h}}^{(q)}_0]. \quad [17.25]$$

The attention vector is computed as a softmax over a vector of bilinear products, and the expected representation is computed by summing over attention values,

$$\tilde{\alpha}_m = (\mathbf{u}^{(q)})^\top \mathbf{W}_a \mathbf{h}_m^{(p)} \quad [17.26]$$

$$\boldsymbol{\alpha} = \text{SoftMax}(\tilde{\boldsymbol{\alpha}}) \quad [17.27]$$

$$\mathbf{o} = \sum_{m=1}^M \alpha_m \mathbf{h}_m^{(p)}. \quad [17.28]$$

Each candidate answer  $c$  is represented by a vector  $\mathbf{x}_c$ . Assuming the candidate answers are spans from the original text, these vectors can be set equal to the corresponding element in  $\mathbf{h}^{(p)}$ . The score for each candidate answer  $a$  is computed by the inner product,

$$\hat{c} = \underset{c}{\operatorname{argmax}} \mathbf{o} \cdot \mathbf{x}_c. \quad [17.29]$$

9369 This architecture can be trained end-to-end from a loss based on the log-likelihood of the  
 9370 correct answer. A number of related architectures have been proposed (e.g., Hermann  
 9371 et al., 2015; Kadlec et al., 2016; Dhingra et al., 2017; Cui et al., 2017), and the relationships  
 9372 between these methods are surveyed by Wang et al. (2017).

### 9373 Additional resources

9374 The field of information extraction is surveyed in course notes by Grishman (2012), and  
 9375 more recently in a short survey paper (Grishman, 2015). Shen et al. (2015) survey the task  
 9376 of entity linking, and Ji and Grishman (2011) survey work on knowledge base popula-  
 9377 tion. This chapter's discussion of non-propositional meaning was strongly influenced by  
 9378 Morante and Sporleder (2012), who introduced a special issue of the journal *Computational  
 9379 Linguistics* dedicated to recent work on modality and negation.

### 9380 Exercises

9381 1. Consider the following heuristic for entity linking:

- 9382 • Among all entities that have the same type as the mention (e.g., LOC, PER),  
 9383 choose the one whose name has the lowest edit distance from the mention.
- 9384 • If more than one entity has the right type and the lowest edit distance from the  
 9385 mention, choose the most popular one.
- 9386 • If no candidate entity has the right type, choose NIL.

Now suppose you have the following feature function:

$$f(y, \mathbf{x}) = [\text{edit-dist}(\text{name}(y), \mathbf{x}), \text{same-type}(y, \mathbf{x}), \text{popularity}(y), \delta(y = \text{NIL})]$$

Design a set of ranking weights  $\theta$  that match the heuristic. You may assume that edit distance and popularity are always in the range [0, 100], and that the NIL entity has values of zero for all features except  $\delta$  ( $y = \text{NIL}$ ).

2. Now consider another heuristic:

- Among all candidate entities that have edit distance zero from the mention and the right type, choose the most popular one.
- If no entity has edit distance zero from the mention, choose the one with the right type that is most popular, regardless of edit distance.
- If no entity has the right type, choose NIL.

Using the same features and assumptions from the previous problem, prove that there is no set of weights that could implement this heuristic. Then show that the heuristic can be implemented by adding a single feature. Your new feature should consider only the edit distance.

3. \* Consider the following formulation for collective entity linking, which rewards sets of entities that are all of the same type, where “types” can be elements of any set:

$$\psi_c(\mathbf{y}) = \begin{cases} \alpha & \text{all entities in } \mathbf{y} \text{ have the same type} \\ \beta & \text{more than half of the entities in } \mathbf{y} \text{ have the same type} \\ 0 & \text{otherwise.} \end{cases} \quad [17.30]$$

Show how to implement this model of collective entity linking in an **integer linear program**. You may want to review § 13.2.2.

To get started, here is an integer linear program for entity linking, without including the collective term  $\psi_c$ :

$$\begin{aligned} \max_{z_{i,y} \in \{0,1\}} \quad & \sum_{i=1}^N \sum_{y \in \mathcal{Y}(\mathbf{x}^{(i)})} s_{i,y} z_{i,y} \\ \text{s.t.} \quad & \sum_{y \in \mathcal{Y}(\mathbf{x}^{(i)})} z_{i,y} \leq 1 \quad \forall i \in \{1, 2, \dots, N\} \end{aligned}$$

where  $z_{i,y} = 1$  if entity  $y$  is linked to mention  $i$ , and  $s_{i,y}$  is a parameter that scores the quality of this individual ranking decision, e.g.,  $s_{i,y} = \theta \cdot f(y, \mathbf{x}^{(i)}, \mathbf{c}^{(i)})$ .

To incorporate the collective linking score, you may assume parameters  $r$ ,

$$r_{y,\tau} = \begin{cases} 1, & \text{entity } y \text{ has type } \tau \\ 0, & \text{otherwise.} \end{cases} \quad [17.31]$$

**Hint:** You will need to define several auxiliary variables to optimize over.

- 9409 4. Run `nltk.corpus.download('reuters')` to download the Reuters corpus in  
 9410 NLTK, and run `from nltk.corpus import reuters` to import it. The com-  
 9411 mand `reuters.words()` returns an iterator over the tokens in the corpus.
- 9412 a) Apply the pattern *\_\_\_\_\_, such as \_\_\_\_\_* to this corpus, obtaining candidates for the  
 9413 IS-A relation, e.g. `IS-A(ROMANIA, COUNTRY)`. What are three pairs that this  
 9414 method identifies correctly? What are three different pairs that it gets wrong?
- 9415 b) Design a pattern for the PRESIDENT relation, e.g. `PRESIDENT(PHILIPPINES, CORAZON AQUINO)`.  
 9416 In this case, you may want to augment your pattern matcher with the ability  
 9417 to match multiple token wildcards, perhaps using case information to detect  
 9418 proper names. Again, list three correct
- 9419 c) Preprocess the Reuters data by running a named entity recognizer, replacing  
 9420 tokens with named entity spans when applicable. Apply your PRESIDENT  
 9421 matcher to this new data. Does the accuracy improve? Compare 20 randomly-  
 9422 selected pairs from this pattern and the one you designed in the previous part.
- 9423 5. Represent the dependency path  $\mathbf{x}^{(i)}$  as a sequence of words and dependency arcs  
 9424 of length  $M_i$ , ignoring the endpoints of the path. In example 1 of Table 17.2, the  
 9425 dependency path is,

$$\mathbf{x}^{(1)} = (\xleftarrow[\text{NSUBJ}]{}, \text{traveled}, \xrightarrow[\text{OBL}]{}) \quad [17.32]$$

9426 If  $x_m^{(i)}$  is a word, then let  $\text{pos}(x_m^{(i)})$  be its part-of-speech, using the tagset defined in  
 9427 chapter 8.

We can define the following kernel function over pairs of dependency paths (Bunescu and Mooney, 2005):

$$\kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \begin{cases} 0, & M_i \neq M_j \\ \prod_{m=1}^{M_i} c(x_m^{(i)}, x_m^{(j)}), & M_i = M_j \end{cases}$$

$$c(x_m^{(i)}, x_m^{(j)}) = \begin{cases} 2, & x_m^{(i)} = x_m^{(j)} \\ 1, & x_m^{(i)} \text{ and } x_m^{(j)} \text{ are words and } \text{pos}(x_m^{(i)}) = \text{pos}(x_m^{(j)}) \\ 0, & \text{otherwise.} \end{cases}$$

9428 Using this kernel function, compute the kernel similarities of example 1 from Ta-  
 9429 ble 17.2 with the other five examples.

6. Continuing from the previous problem, suppose that the instances have the following labels:

$$y_2 = 1, y_3 = -1, y_4 = -1, y_5 = 1, y_6 = 1 \quad [17.33]$$

9430 Identify the conditions for  $\alpha$  and  $b$  under which  $\hat{y}_1 = 1$ . Remember the constraint  
 9431 that  $\alpha_i \geq 0$  for all  $i$ .



## 9432 Chapter 18

# 9433 Machine translation

9434 Machine translation (MT) is one of the “holy grail” problems in artificial intelligence,  
9435 with the potential to transform society by facilitating communication between people  
9436 anywhere in the world. As a result, MT has received significant attention and funding  
9437 since the early 1950s. However, it has proved remarkably challenging, and while there  
9438 has been substantial progress towards usable MT systems — especially for high-resource  
9439 language pairs like English-French — we are still far from translation systems that match  
9440 the nuance and depth of human translations.

### 9441 18.1 Machine translation as a task

9442 Machine translation can be formulated as an optimization problem:

$$\hat{\mathbf{w}}^{(t)} = \underset{\mathbf{w}^{(t)}}{\operatorname{argmax}} \Psi(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}), \quad [18.1]$$

9443 where  $\mathbf{w}^{(s)}$  is a sentence in a **source** language,  $\mathbf{w}^{(t)}$  is a sentence in the **target language**,  
9444 and  $\Psi$  is a scoring function. As usual, this formalism requires two components: a decod-  
9445 ing algorithm for computing  $\hat{\mathbf{w}}^{(t)}$ , and a learning algorithm for estimating the parameters  
9446 of the scoring function  $\Psi$ .

9447 Decoding is difficult for machine translation because of the huge space of possible  
9448 translations. We have faced large label spaces before: for example, in sequence labeling,  
9449 the set of possible label sequences is exponential in the length of the input. In these cases,  
9450 it was possible to search the space quickly by introducing locality assumptions: for ex-  
9451 ample, that each tag depends only on its predecessor, or that each production depends  
9452 only on its parent. In machine translation, no such locality assumptions seem possible:  
9453 human translators reword, reorder, and rearrange words; they replace single words with  
9454 multi-word phrases, and vice versa. This flexibility means that in even relatively simple

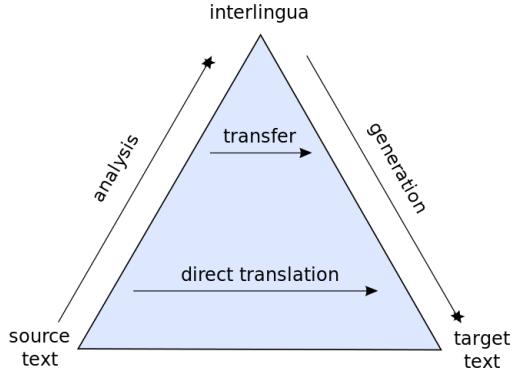


Figure 18.1: The Vauquois Pyramid [http://commons.wikimedia.org/wiki/File:Direct\\_translation\\_and\\_transfer\\_translation\\_pyramind.svg](http://commons.wikimedia.org/wiki/File:Direct_translation_and_transfer_translation_pyramind.svg)

9455 translation models, decoding is NP-hard (Knight, 1999). Approaches for dealing with this  
 9456 complexity are described in § 18.4.

Estimating translation models is difficult as well. Labeled translation data usually comes in the form parallel sentences, e.g.,

$$\begin{aligned} w^{(s)} &= A \text{ } Vinay \text{ } le \text{ } gusta \text{ } las \text{ } manzanas. \\ w^{(t)} &= Vinay \text{ likes apples.} \end{aligned}$$

9457 A useful feature function would note the translation pairs (*gusta, likes*), (*manzanas, apples*),  
 9458 and even (*Vinay, Vinay*). But this word-to-word **alignment** is not given in the data. One  
 9459 solution is to treat this alignment as a **latent variable**; this is the approach taken by clas-  
 9460 **statistical machine translation** (SMT) systems, described in § 18.2. Another solution  
 9461 is to model the relationship between  $w^{(t)}$  and  $w^{(s)}$  through a more complex and expres-  
 9462 sive function; this is the approach taken by **neural machine translation** (NMT) systems,  
 9463 described in § 18.3.

9464 The **Vauquois Pyramid** is a theory of how translation should be done. At the lowest  
 9465 level, the translation system operates on individual words, but the horizontal distance  
 9466 at this level is large, because languages express ideas differently. If we can move up the  
 9467 triangle to syntactic structure, the distance for translation is reduced; we then need only  
 9468 produce target-language text from the syntactic representation, which can be as simple  
 9469 as reading off a tree. Further up the triangle lies semantics; translating between semantic  
 9470 representations should be easier still, but mapping between semantics and surface text is  
 9471 a difficult, unsolved problem. At the top of the triangle is **interlingua**, a semantic repre-  
 9472 **sentation** that is so generic that it is identical across all human languages. Philosophers

	Adequate?	Fluent?
<i>To Vinay it like Python</i>	yes	no
<i>Vinay debugs memory leaks</i>	no	yes
<i>Vinay likes Python</i>	yes	yes

Table 18.1: Adequacy and fluency for translations of the Spanish sentence *A Vinay le gusta Python*.

debate whether such a thing as interlingua is really possible (Derrida, 1985). While the first-order logic representations discussed in chapter 12 might be considered to be language independent, it is built on an inventory of relations that is suspiciously similar to a subset of English words (Nirenburg and Wilks, 2001). Nonetheless, the idea of linking translation and semantic understanding may still be a promising path, if the resulting translations better preserve the meaning of the original text.

### 18.1.1 Evaluating translations

There are two main criteria for a translation, summarized in Table 18.1.

- **Adequacy:** The translation  $w^{(t)}$  should adequately reflect the linguistic content of  $w^{(s)}$ . For example, if  $w^{(s)} = A Vinay le gusta Python$ , the gloss<sup>1</sup>  $w^{(t)} = To Vinay it like Python$  is considered adequate because it contains all the relevant content. The output  $w^{(t)} = Vinay debugs memory leaks$  is not adequate.
- **Fluency:** The translation  $w^{(t)}$  should read like fluent text in the target language. By this criterion, the gloss  $w^{(t)} = To Vinay it like Python$  will score poorly, and  $w^{(t)} = Vinay debugs memory leaks$  will be preferred.

Automated evaluations of machine translations typically merge both of these criteria, by comparing the system translation with one or more **reference translations**, produced by professional human translators. The most popular quantitative metric is **BLEU** (bilingual evaluation understudy; Papineni et al., 2002), which is based on  $n$ -gram precision: what fraction of  $n$ -grams in the system translation appear in the reference? Specifically, for each  $n$ -gram length, the precision is defined as,

$$p_n = \frac{\text{number of } n\text{-grams appearing in both reference and hypothesis translations}}{\text{number of } n\text{-grams appearing in the hypothesis translation}}. \quad [18.2]$$

The  $n$ -gram precisions for three hypothesis translations are shown in Figure 18.2.

---

<sup>1</sup>A gloss is a word-for-word translation.

	<b>Translation</b>	$p_1$	$p_2$	$p_3$	$p_4$	BP	BLEU
<i>Reference</i>	<i>Vinay likes programming in Python</i>						
<i>Sys1</i>	<i>To Vinay it like to program Python</i>	$\frac{2}{7}$	0	0	0	1	.21
<i>Sys2</i>	<i>Vinay likes Python</i>	$\frac{3}{3}$	$\frac{1}{2}$	0	0	.51	.33
<i>Sys3</i>	<i>Vinay likes programming in his pajamas</i>	$\frac{4}{6}$	$\frac{3}{5}$	$\frac{2}{4}$	$\frac{1}{3}$	1	.76

Figure 18.2: A reference translation and three system outputs. For each output,  $p_n$  indicates the precision at each  $n$ -gram, and BP indicates the brevity penalty.

9495     The BLEU score is then based on the average,  $\exp \frac{1}{N} \sum_{n=1}^N \log p_n$ . Two modifications  
 9496     of Equation 18.2 are necessary: (1) to avoid computing  $\log 0$ , all precisions are smoothed  
 9497     to ensure that they are positive; (2) each  $n$ -gram in the source can be used at most once,  
 9498     so that *to to to to to* does not achieve  $p_1 = 1$  against the reference *to be or not to be*.  
 9499     Furthermore, precision-based metrics are biased in favor of short translations, which can  
 9500     achieve high scores by minimizing the denominator in [18.2]. To avoid this issue, a **brevity**  
 9501     **penalty** is applied to translations that are shorter than the reference. This penalty is indi-  
 9502     cated as “BP” in Figure 18.2.

9503     Automated metrics like BLEU have been validated by correlation with human judg-  
 9504     ments of translation quality. Nonetheless, it is not difficult to construct examples in which  
 9505     the BLEU score is high, yet the translation is disfluent or carries a completely different  
 9506     meaning from the original. To give just one example, consider the problem of translating  
 9507     pronouns. Because pronouns refer to specific entities, a single incorrect pronoun can obliti-  
 9508     onate the semantics of the original sentence. Existing state-of-the-art systems generally  
 9509     do not attempt the reasoning necessary to correctly resolve pronominal anaphora (Hard-  
 9510     meier, 2012). Despite the importance of pronouns for semantics, they have a marginal  
 9511     impact on BLEU, which may help to explain why existing systems do not make a greater  
 9512     effort to translate them correctly.

9513     **Fairness and bias**     The problem of pronoun translation intersects with issues of fairness  
 9514     and bias. In many languages, such as Turkish, the third person singular pronoun is gender  
 9515     neutral. Today’s state-of-the-art systems produce the following Turkish-English transla-  
 9516     tions (Caliskan et al., 2017):

- 9517     (18.1) *O bir doktor.*  
                 He is a doctor.  
 9518     (18.2) *O bir hemşire.*  
                 She is a nurse.

9519 The same problem arises for other professions that have stereotypical genders, such as  
9520 engineers, soldiers, and teachers, and for other languages that have gender-neutral pro-  
9521 nouns. This bias was not directly programmed into the translation model; it arises from  
9522 statistical tendencies in existing datasets. This highlights a general problem with data-  
9523 driven approaches, which can perpetuate biases that negatively impact disadvantaged  
9524 groups. Worse, machine learning can *amplify* biases in data (Bolukbasi et al., 2016): if a  
9525 dataset has even a slight tendency towards men as doctors, the resulting translation model  
9526 may produce translations in which doctors are always *he*, and nurses are always *she*.

9527 **Other metrics** A range of other automated metrics have been proposed for machine  
9528 translation. One potential weakness of BLEU is that it only measures precision; METEOR  
9529 is a weighted *F*-MEASURE, which is a combination of recall and precision (see § 4.4.1).  
9530 **Translation Error Rate (TER)** computes the string **edit distance** (see § 9.1.4.1) between the  
9531 reference and the hypothesis (Snover et al., 2006). For language pairs like English and  
9532 Japanese, there are substantial differences in word order, and word order errors are not  
9533 sufficiently captured by *n*-gram based metrics. The **RIBES** metric applies rank corre-  
9534 lation to measure the similarity in word order between the system and reference transla-  
9535 tions (Isozaki et al., 2010).

### 9536 18.1.2 Data

9537 Data-driven approaches to machine translation rely primarily on **parallel corpora**: sentence-  
9538 level translations. Early work focused on government records, in which fine-grained offi-  
9539 cial translations are often required. For example, the IBM translation systems were based  
9540 on the proceedings of the Canadian Parliament, called **Hansards**, which are recorded in  
9541 English and French (Brown et al., 1990). The growth of the European Union led to the  
9542 development of the **EuroParl corpus**, which spans 21 European languages (Koehn, 2005).  
9543 While these datasets helped to launch the field of machine translation, they are restricted  
9544 to narrow domains and a formal speaking style, limiting their applicability to other types  
9545 of text. As more resources are committed to machine translation, new translation datasets  
9546 have been commissioned. This has broadened the scope of available data to news,<sup>2</sup> movie  
9547 subtitles,<sup>3</sup> social media (Ling et al., 2013), dialogues (Fordyce, 2007), TED talks (Paul et al.,  
9548 2010), and scientific research articles (Nakazawa et al., 2016).

9549 Despite this growing set of resources, the main bottleneck in machine translation data  
9550 is the need for parallel corpora that are aligned at the sentence level. Many languages have  
9551 sizable parallel corpora with some high-resource language, but not with each other. The  
9552 high-resource language can then be used as a “pivot” or “bridge” (Boitet, 1988; Utiyama

---

<sup>2</sup>[https://catalog.ldc.upenn.edu/LDC2010T10\\_translation-task.html](https://catalog.ldc.upenn.edu/LDC2010T10_translation-task.html) <sup>3</sup><http://www.statmt.org/wmt15/>

<sup>3</sup><http://opus.nlpl.eu/>

and Isahara, 2007): for example, De Gispert and Marino (2006) use Spanish as a bridge for translation between Catalan and English. For most of the 6000 languages spoken today, the only source of translation data remains the Judeo-Christian Bible (Resnik et al., 1999). While relatively small, at less than a million tokens, the Bible has been translated into more than 2000 languages, far outpacing any other corpus. Some research has explored the possibility of automatically identifying parallel sentence pairs from unaligned parallel texts, such as web pages and Wikipedia articles (Kilgarriff and Grefenstette, 2003; Resnik and Smith, 2003; Adafre and De Rijke, 2006). Another approach is to create large parallel corpora through crowdsourcing (Zaidan and Callison-Burch, 2011).

## 18.2 Statistical machine translation

The previous section introduced adequacy and fluency as the two main criteria for machine translation. A natural modeling approach is to represent them with separate scores,

$$\Psi(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \Psi_A(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) + \Psi_F(\mathbf{w}^{(t)}). \quad [18.3]$$

The fluency score  $\Psi_F$  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

	<i>A</i>	<i>Vinay</i>	<i>le</i>	<i>gusta</i>	<i>python</i>
<i>Vinay</i>		■			
<i>likes</i>			■	■	
<i>python</i>					■

Figure 18.3: An example word-to-word alignment

9581  $p_{T|S}$ . Under this interpretation, the target probability  $p_T$  is just a language model, and  
 9582 can be estimated using any of the techniques from chapter 6. The only remaining learning  
 9583 problem is to estimate the translation model  $p_{S|T}$ .

### 9584 18.2.1 Statistical translation modeling

9585 The simplest decomposition of the translation model is word-to-word: each word in the  
 9586 source should be aligned to a word in the translation. This approach presupposes an  
 9587 **alignment**  $\mathcal{A}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)})$ , which contains a list of pairs of source and target tokens. For  
 9588 example, given  $\mathbf{w}^{(s)} = A\ Vinay\ le\ gusta\ Python$  and  $\mathbf{w}^{(t)} = Vinay\ likes\ Python$ , one possible  
 9589 word-to-word alignment is,

$$\mathcal{A}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \{(A, \emptyset), (Vinay, Vinay), (le, likes), (gusta, likes), (Python, Python)\}. \quad [18.7]$$

9590 This alignment is shown in Figure 18.3. Another, less promising, alignment is:

$$\mathcal{A}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \{(A, Vinay), (Vinay, likes), (le, Python), (gusta, \emptyset), (Python, \emptyset)\}. \quad [18.8]$$

9591 Each alignment contains exactly one tuple for each word in the *source*, which serves to  
 9592 explain how the source word could be translated from the target, as required by the trans-  
 9593 lation probability  $p_{S|T}$ . If no appropriate word in the target can be identified for a source  
 9594 word, it is aligned to  $\emptyset$  — as is the case for the Spanish function word *a* in the example,  
 9595 which glosses to the English word *to*. Words in the target can align with multiple words  
 9596 in the source, so that the target word *likes* can align to both *le* and *gusta* in the source.

The joint probability of the alignment and the translation can be defined conveniently

as,

$$p(\mathbf{w}^{(s)}, \mathcal{A} | \mathbf{w}^{(t)}) = \prod_{m=1}^{M^{(s)}} p(w_m^{(s)}, a_m | w_{a_m}^{(t)}, m, M^{(s)}, M^{(t)}) \quad [18.9]$$

$$= \prod_{m=1}^{M^{(s)}} p(a_m | m, M^{(s)}, M^{(t)}) \times p(w_m^{(s)} | w_{a_m}^{(t)}). \quad [18.10]$$

9597 This probability model makes two key assumptions:

9598 • The alignment probability factors across tokens,

$$p(\mathcal{A} | \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \prod_{m=1}^{M^{(s)}} p(a_m | m, M^{(s)}, M^{(t)}). \quad [18.11]$$

9599 This means that each alignment decision is independent of the others, and depends  
9600 only on the index  $m$ , and the sentence lengths  $M^{(s)}$  and  $M^{(t)}$ .

9601 • The translation probability also factors across tokens,

$$p(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}, \mathcal{A}) = \prod_{m=1}^{M^{(s)}} p(w_m^{(s)} | w_{a_m}^{(t)}), \quad [18.12]$$

9602 so that each word in  $\mathbf{w}^{(s)}$  depends only on its aligned word in  $\mathbf{w}^{(t)}$ . This means that  
9603 translation is word-to-word, ignoring context. The hope is that the target language  
9604 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:

$$p(a_m | m, M^{(s)}, M^{(t)}) = \frac{1}{M^{(t)}}. \quad [18.16]$$

9605 In this model, every alignment is equally likely. This is almost surely wrong, but it re-  
 9606 sults in a convex learning objective, yielding a good initialization for the more complex  
 9607 alignment models (Brown et al., 1993; Koehn, 2009).

### 9608 18.2.2 Estimation

9609 Let us define the parameter  $\theta_{u \rightarrow v}$  as the probability of translating target word  $u$  to source  
 9610 word  $v$ . If word-to-word alignments were annotated, these probabilities could be com-  
 9611 puted from relative frequencies,

$$\hat{\theta}_{u \rightarrow v} = \frac{\text{count}(u, v)}{\text{count}(u)}, \quad [18.17]$$

9612 where  $\text{count}(u, v)$  is the count of instances in which word  $v$  was aligned to word  $u$  in  
 9613 the training set, and  $\text{count}(u)$  is the total count of the target word  $u$ . The smoothing  
 9614 techniques mentioned in chapter 6 can help to reduce the variance of these probability  
 9615 estimates.

9616 Conversely, if we had an accurate translation model, we could estimate the likelihood  
 9617 of each alignment decision,

$$q_m(a_m \mid \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) \propto p(a_m \mid m, M^{(s)}, M^{(t)}) \times p(w_m^{(s)} \mid w_{a_m}^{(t)}), \quad [18.18]$$

where  $q_m(a_m \mid \mathbf{w}^{(s)}, \mathbf{w}^{(t)})$  is a measure of our confidence in aligning source word  $w_m^{(s)}$   
 to target word  $w_{a_m}^{(t)}$ . The relative frequencies could then be computed from the *expected  
 counts*,

$$\hat{\theta}_{u \rightarrow v} = \frac{E_q[\text{count}(u, v)]}{\text{count}(u)} \quad [18.19]$$

$$E_q[\text{count}(u, v)] = \sum_m q_m(a_m \mid \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) \delta(w_m^{(s)} = v) \delta(w_{a_m}^{(t)} = u). \quad [18.20]$$

9618 The **expectation-maximization** (EM) algorithm proceeds by iteratively updating  $q_m$   
 9619 and  $\hat{\Theta}$ . The algorithm is described in general form in chapter 5. For statistical machine  
 9620 translation, the steps of the algorithm are:

- 9621 1. **E-step:** Update beliefs about word alignment using Equation 18.18.
- 9622 2. **M-step:** Update the translation model using Equations 18.19 and 18.20.

9623 As discussed in chapter 5, the expectation maximization algorithm is guaranteed to con-  
 9624 verge, but not to a global optimum. However, for IBM Model 1, it can be shown that EM  
 9625 optimizes a convex objective, and global optimality is guaranteed. For this reason, IBM  
 9626 Model 1 is often used as an initialization for more complex alignment models. For more  
 9627 detail, see Koehn (2009).

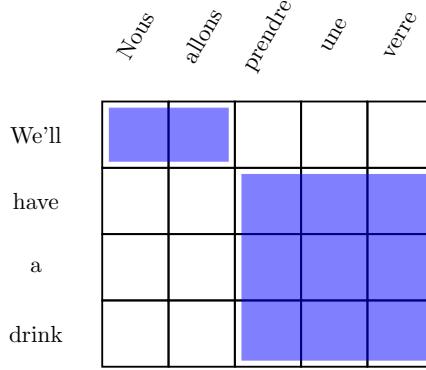


Figure 18.4: A phrase-based alignment between French and English, corresponding to example (18.3)

### 18.2.3 Phrase-based translation

Real translations are not word-to-word substitutions. One reason is that many multiword expressions are not translated literally, as shown in this example from French:

- (18.3) *Nous allons prendre un verre*  
 We will take a glass  
 We'll have a drink

The line *we will take a glass* is the word-for-word gloss of the French sentence; the translation *we'll have a drink* is shown on the third line. Such examples are difficult for word-to-

word translation models, since they require translating *prendre* to *have* and *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

$$p(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}, \mathcal{A}) = \prod_{((i, j), (k, \ell)) \in \mathcal{A}} p_{w^{(s)}|w^{(t)}}(\{w_{i+1}^{(s)}, w_{i+2}^{(s)}, \dots, w_j^{(s)}\} | \{w_{k+1}^{(t)}, w_{k+2}^{(t)}, \dots, w_\ell^{(t)}\}). \quad [18.21]$$

The phrase alignment  $((i, j), (k, \ell))$  indicates that the span  $\mathbf{w}_{i+1:j}^{(s)}$  is the translation of the span  $\mathbf{w}_{k+1:\ell}^{(t)}$ . An example phrasal alignment is shown in Figure 18.4. Note that the align-

ment set  $\mathcal{A}$  is required to cover all of the tokens in the source, just as in word-based translation. The probability model  $p_{w^{(s)}|w^{(t)}}$  must now include translations for all phrase pairs, which can be learned from expectation-maximization just as in word-based statistical machine translation.

### 18.2.4 \*Syntax-based translation

The Vauquois Pyramid (Figure 18.1) suggests that translation might be easier if we take a higher-level view. One possibility is to incorporate the syntactic structure of the source, the target, or both. This is particularly promising for language pairs that consistent syntactic differences. For example, English adjectives almost always precede the nouns that they modify, while in Romance languages such as French and Spanish, the adjective often follows the noun: thus, *angry fish* would translate to *pez (fish) enojado (angry)* in Spanish. In word-to-word translation, these reorderings cause the alignment model to be overly permissive. It is not that the order of *any* pair of English words can be reversed when translating into Spanish, but only adjectives and nouns within a noun phrase. Similar issues arise when translating between verb-final languages such as Japanese (in which verbs usually follow the subject and object), verb-initial languages like Tagalog and classical Arabic, and verb-medial languages such as English.

An elegant solution is to link parsing and translation in a **synchronous context-free grammar** (SCFG; Chiang, 2007).<sup>4</sup> An SCFG is a set of productions of the form  $X \rightarrow (\alpha, \beta, \sim)$ , where  $X$  is a non-terminal,  $\alpha$  and  $\beta$  are sequences of terminals or non-terminals, and  $\sim$  is a one-to-one alignment of items in  $\alpha$  with items in  $\beta$ . To handle the English-Spanish adjective-noun ordering, an SCFG would include productions such as,

$$\text{NP} \rightarrow (\text{DET}_1 \text{NN}_2 \text{JJ}_3, \quad \text{DET}_1 \text{JJ}_3 \text{NN}_2), \quad [18.22]$$

with subscripts indicating the alignment between the Spanish (left) and English (right) parts of the right-hand side. Terminal productions yield translation pairs,

$$\text{JJ} \rightarrow (\text{enojado}_1, \text{angry}_1). \quad [18.23]$$

A synchronous derivation begins with the start symbol  $S$ , and derives a pair of sequences of terminal symbols.

Given an SCFG in which each production yields at most two symbols in each language (Chomsky Normal Form; see § 9.2.1.2), a sentence can be parsed using only the CKY algorithm (chapter 10). The resulting derivation also includes productions in the other language, all the way down to the surface form. Therefore, SCFGs make translation very similar to parsing. In a weighted SCFG, the log probability  $\log p_{S|T}$  can be computed from

---

<sup>4</sup>Key earlier work includes syntax-driven transduction (Lewis II and Stearns, 1968) and stochastic inversion transduction grammars (Wu, 1997).

the sum of the log-probabilities of the productions. However, combining SCFGs with a target language model is computationally expensive, necessitating approximate search algorithms (Huang and Chiang, 2007).

Synchronous context-free grammars are an example of **tree-to-tree translation**, because they model the syntactic structure of both the target and source language. In **string-to-tree translation**, string elements are translated into constituent tree fragments, which are then assembled into a translation (Yamada and Knight, 2001; Galley et al., 2004); in **tree-to-string translation**, the source side is parsed, and then transformed into a string on the target side (Liu et al., 2006). A key question for syntax-based translation is the extent to which we phrasal constituents align across translations (Fox, 2002), because this governs the extent to which we can rely on monolingual parsers and treebanks. For more on syntax-based machine translation, see the monograph by Williams et al. (2016).

### 18.3 Neural machine translation

Neural network models for machine translation are based on the **encoder-decoder** architecture (Cho et al., 2014). The encoder network converts the source language sentence into a vector or matrix representation; the decoder network then converts the encoding into a sentence in the target language.

$$\mathbf{z} = \text{ENCODE}(\mathbf{w}^{(s)}) \quad [18.24]$$

$$\mathbf{w}^{(t)} \mid \mathbf{w}^{(s)} \sim \text{DECODE}(\mathbf{z}), \quad [18.25]$$

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,

$$w_m^{(t)} \mid \mathbf{w}_{1:m-1}^{(t)}, \mathbf{w}^{(s)} \sim \text{SoftMax} \left( \boldsymbol{\beta} \cdot \mathbf{h}_{m-1}^{(t)} \right), \quad [18.28]$$

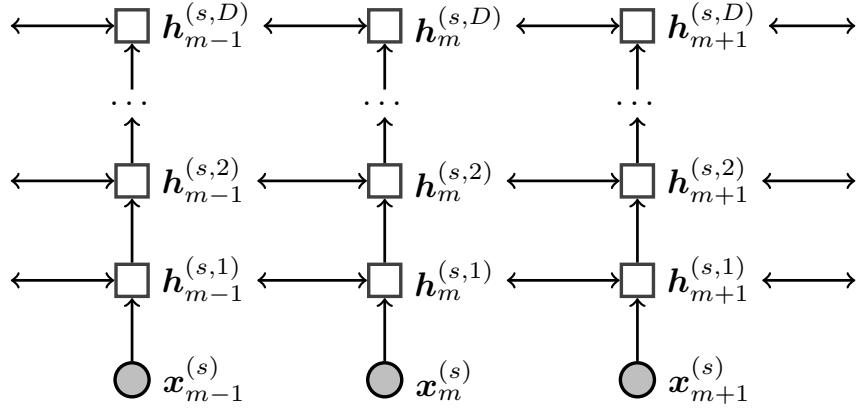


Figure 18.5: A deep bidirectional LSTM encoder

9685 where  $\beta \in \mathbb{R}^{(V^{(t)} \times K)}$  is the matrix of output word vectors for the  $V^{(t)}$  words in the target  
 9686 language vocabulary.

The simplest encoder-decoder architecture is the **sequence-to-sequence** model (Sutskever et al., 2014). In this model, the encoder is set to the final hidden state of a **long short-term memory (LSTM)** (see § 6.3.3) on the source sentence:

$$\mathbf{h}_m^{(s)} = \text{LSTM}(\mathbf{x}_m^{(s)}, \mathbf{h}_{m-1}^{(s)}) \quad [18.29]$$

$$\mathbf{z} \triangleq \mathbf{h}_{M^{(s)}}^{(s)}, \quad [18.30]$$

where  $\mathbf{x}_m^{(s)}$  is the embedding of source language word  $w_m^{(s)}$ . The encoding then provides the initial hidden state for the decoder LSTM:

$$\mathbf{h}_0^{(t)} = \mathbf{z} \quad [18.31]$$

$$\mathbf{h}_m^{(t)} = \text{LSTM}(\mathbf{x}_m^{(t)}, \mathbf{h}_{m-1}^{(t)}), \quad [18.32]$$

9687 where  $\mathbf{x}_m^{(t)}$  is the embedding of the target language word  $w_m^{(t)}$ .

9688 Sequence-to-sequence translation is nothing more than wiring together two LSTMs:  
 9689 one to read the source, and another to generate the target. To make the model work well,  
 9690 some additional tweaks are needed:

- 9691 • Most notably, the model works much better if the source sentence is reversed, reading  
 9692 from the end of the sentence back to the beginning. In this way, the words at the  
 9693 beginning of the source have the greatest impact on the encoding  $\mathbf{z}$ , and therefore

9694 impact the words at the beginning of the target sentence. Later work on more ad-  
 9695 vanced encoding models, such as **neural attention** (see § 18.3.1), has eliminated the  
 9696 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  $i + 1$ :

$$\mathbf{h}_m^{(s,1)} = \text{LSTM}(\mathbf{x}_m^{(s)}, \mathbf{h}_{m-1}^{(s)}) \quad [18.33]$$

$$\mathbf{h}_m^{(s,i+1)} = \text{LSTM}(\mathbf{h}_m^{(s,i)}, \mathbf{h}_{m-1}^{(s)}), \quad \forall i \geq 1. \quad [18.34]$$

9697 The original work on sequence-to-sequence translation used four layers; in 2016,  
 9698 Google’s commercial machine translation system used eight layers (Wu et al., 2016).<sup>5</sup>

- 9699 • Significant improvements can be obtained by creating an **ensemble** of translation  
 9700 models, each trained from a different random initialization. For an ensemble of size  
 9701  $N$ , the per-token decoding probability is set equal to,

$$p(w^{(t)} | \mathbf{z}, \mathbf{w}_{1:m-1}^{(t)}) = \frac{1}{N} \sum_{i=1}^N p_i(w^{(t)} | \mathbf{z}, \mathbf{w}_{1:m-1}^{(t)}), \quad [18.35]$$

9702 where  $p_i$  is the decoding probability for model  $i$ . Each translation model in the  
 9703 ensemble includes its own encoder and decoder networks.

- 9704 • The original sequence-to-sequence model used a fairly standard training setup: stochas-  
 9705 tic gradient descent with an exponentially decreasing learning rate after the first five  
 9706 epochs; mini-batches of 128 sentences, chosen to have similar length so that each  
 9707 sentence on the batch will take roughly the same amount of time to process; gradi-  
 9708 ent clipping (see § 3.3.4) to ensure that the norm of the gradient never exceeds some  
 9709 predefined value.

### 9710 18.3.1 Neural attention

9711 The sequence-to-sequence model discussed in the previous section was a radical depart-  
 9712 ure from statistical machine translation, in which each word or phrase in the target lan-  
 9713 guage is conditioned on a single word or phrase in the source language. Both approaches  
 9714 have advantages. Statistical translation leverages the idea of compositionality — transla-  
 9715 tions of large units should be based on the translations of their component parts — and  
 9716 this seems crucial if we are to scale translation to longer units of text. But the translation  
 9717 of each word or phrase often depends on the larger context, and encoder-decoder models  
 9718 capture this context at the sentence level.

---

<sup>5</sup>Google reports that this system took six days to train for English-French translation, using 96 NVIDIA K80 GPUs, which would have cost roughly half a million dollars at the time.

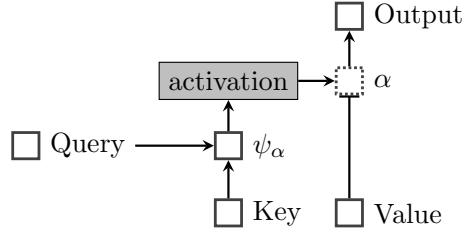


Figure 18.6: A general view of neural attention. The dotted box indicates that each  $\alpha_{m \rightarrow n}$  can be viewed as a **gate** on value  $n$ .

Is it possible for translation to be both contextualized and compositional? One approach is to augment neural translation with an **attention mechanism**. The idea of neural attention was described in § 17.5, but its application to translation bears further discussion. In general, attention can be thought of as using a query to select from a memory of key-value pairs. However, the query, keys, and values are all vectors, and the entire operation is differentiable. For each key  $n$  in the memory, we compute a score  $\psi_\alpha(m, n)$  with respect to the query  $m$ . That score is a function of the compatibility of the key and the query, and can be computed using a small feedforward neural network. The vector of scores is passed through an activation function, such as softmax. The output of this activation function is a vector of non-negative numbers  $[\alpha_{m \rightarrow 1}, \alpha_{m \rightarrow 2}, \dots, \alpha_{m \rightarrow N}]^\top$ , with length  $N$  equal to the size of the memory. Each value in the memory  $v_n$  is multiplied by the attention  $\alpha_{m \rightarrow n}$ ; the sum of these scaled values is the output. This process is shown in Figure 18.6. In the extreme case that  $\alpha_{m \rightarrow n} = 1$  and  $\alpha_{m \rightarrow n'} = 0$  for all other  $n'$ , then the attention mechanism simply selects the value  $v_n$  from the memory.

Neural attention makes it possible to integrate alignment into the encoder-decoder architecture. Rather than encoding the entire source sentence into a fixed length vector  $z$ , it can be encoded into a matrix  $Z \in \mathbb{R}^{K \times M^{(S)}}$ , where  $K$  is the dimension of the hidden state, and  $M^{(S)}$  is the number of tokens in the source input. Each column of  $Z$  represents the state of a recurrent neural network over the source sentence. These vectors are constructed from a **bidirectional LSTM** (see § 7.6), which can be a deep network as shown in Figure 18.5. These columns are both the keys and the values in the attention mechanism.

At each step  $m$  in decoding, the attentional state is computed by executing a query, which is equal to the state of the decoder,  $h_m^{(t)}$ . The resulting compatibility scores are,

$$\psi_\alpha(m, n) = v_\alpha \cdot \tanh(\Theta_\alpha[h_m^{(t)}; h_n^{(s)}]). \quad [18.36]$$

The function  $\psi$  is thus a two layer feedforward neural network, with weights  $v_\alpha$  on the output layer, and weights  $\Theta_\alpha$  on the input layer. To convert these scores into attention weights, we apply an activation function, which can be vector-wise softmax or an

9743 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  $\alpha$  is then used to compute an **context vector**  $c_m$  by taking a weighted average over the columns of  $Z$ ,

$$c_m = \sum_{n=1}^{M^{(s)}} \alpha_{m \rightarrow n} z_n, \quad [18.39]$$

where  $\alpha_{m \rightarrow n} \in [0, 1]$  is the amount of attention from word  $m$  of the target to word  $n$  of the source. The context vector can be incorporated into the decoder's word output probability model, by adding another layer to the decoder (Luong et al., 2015):

$$\tilde{h}_m^{(t)} = \tanh(\Theta_c[h_m^{(t)}; c_m]) \quad [18.40]$$

$$p(w_{m+1}^{(t)} | w_{1:m}^{(t)}, w^{(s)}) \propto \exp\left(\beta_{w_{m+1}^{(t)}} \cdot \tilde{h}_m^{(t)}\right). \quad [18.41]$$

9744 Here the decoder state  $h_m^{(t)}$  is concatenated with the context vector, forming the input  
 9745 to compute a final output vector  $\tilde{h}_m^{(t)}$ . The context vector can be incorporated into the  
 9746 decoder recurrence in a similar manner (Bahdanau et al., 2014).

9747 **18.3.2 \*Neural machine translation without recurrence**

In the encoder-decoder model, attention's “keys and values” are the hidden state representations in the encoder network,  $z$ , and the “queries” are state representations in the decoder network  $h^{(t)}$ . It is also possible to completely eliminate recurrence from neural translation, by applying **self-attention** (Lin et al., 2017; Kim et al., 2017) within the encoder and decoder, as in the **transformer architecture** (Vaswani et al., 2017). For level  $i$ , the basic equations of the encoder side of the transformer are:

$$z_m^{(i)} = \sum_{n=1}^{M^{(s)}} \alpha_{m \rightarrow n}^{(i)} (\Theta_v h_n^{(i-1)}) \quad [18.42]$$

$$h_m^{(i)} = \Theta_2 \text{ReLU}(\Theta_1 z_m^{(i)} + b_1) + b_2. \quad [18.43]$$

9748 For each token  $m$  at level  $i$ , we compute self-attention over the entire source sentence:  
 9749 the keys, values, and queries are all projections of the vector  $\mathbf{h}^{(i-1)}$ . The attention scores  
 9750  $\alpha_{m \rightarrow n}^{(i)}$  are computed using a scaled form of softmax attention,

$$\alpha_{m \rightarrow n} \propto \exp(\psi_\alpha(m, n)/M), \quad [18.44]$$

9751 where  $M$  is the length of the input. This encourages the attention to be more evenly  
 9752 dispersed across the input. Self-attention is applied across multiple “heads”, each using  
 9753 different projections of  $\mathbf{h}^{(i-1)}$  to form the keys, values, and queries.

9754 The output of the self-attentional layer is the representation  $\mathbf{z}_m^{(i)}$ , which is then passed  
 9755 through a two-layer feed-forward network, yielding the input to the next layer,  $\mathbf{h}^{(i)}$ . To  
 9756 ensure that information about word order in the source is integrated into the model, the  
 9757 encoder includes **positional encodings** of the index of each word in the source. These  
 9758 encodings are vectors for each position  $m \in \{1, 2, \dots, M\}$ . The positional encodings are  
 9759 concatenated with the word embeddings  $\mathbf{x}_m$  at the base layer of the model.<sup>6</sup>

9760 Convolutional neural networks (see § 3.4) have also been applied as encoders in neu-  
 9761 ral machine translation. For each word  $w_m^{(s)}$ , a convolutional network computes a rep-  
 9762 resentation  $\mathbf{h}_m^{(s)}$  from the embeddings of the word and its neighbors. This procedure is  
 9763 applied several times, creating a deep convolutional network. The recurrent decoder then  
 9764 computes a set of attention weights over these convolutional representations, using the  
 9765 decoder’s hidden state  $\mathbf{h}^{(t)}$  as the queries. This attention vector is used to compute a  
 9766 weighted average over the outputs of *another* convolutional neural network of the source,  
 9767 yielding an averaged representation  $\mathbf{c}_m$ , which is then fed into the decoder. As with the  
 9768 transformer, speed is the main advantage over recurrent encoding models; another sim-  
 9769 ilarity is that word order information is approximated through the use of positional en-  
 9770 codings. It seems likely that there are limitations to how well positional encodings can  
 9771 account for word order and deeper linguistic structure. But for the moment, the com-  
 9772 putational advantages of such approaches have put them on par with the best recurrent  
 9773 translation models.<sup>7</sup>

### 9774 18.3.3 Out-of-vocabulary words

9775 Thus far, we have treated translation as a problem at the level of words or phrases. For  
 9776 words that do not appear in the training data, all such models will struggle. There are  
 9777 two main reasons for the presence of out-of-vocabulary (OOV) words:

---

<sup>6</sup>The 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).

<sup>7</sup>A 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.

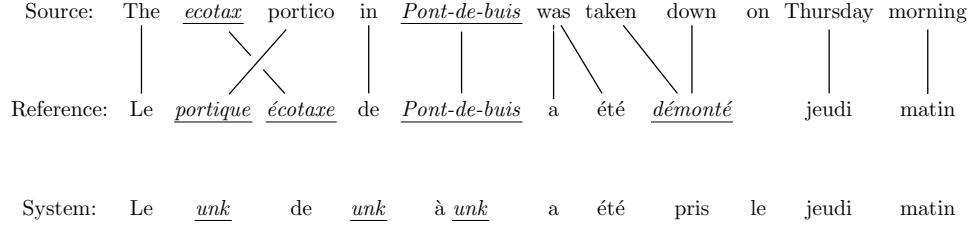


Figure 18.7: Translation with *unknown words*. The system outputs *unk* to indicate words that are outside its vocabulary. Figure adapted from Luong et al. (2015).

- New proper nouns, such as family names or organizations, are constantly arising — particularly in the news domain. The same is true, to a lesser extent, for technical terminology. This issue is shown in Figure 18.7.
- In many languages, words have complex internal structure, known as **morphology**. An example is German, which uses compounding to form nouns like *Abwasserbehandlungsanlage* (*sewage water treatment plant*; example from Sennrich et al. (2016)). While compounds could in principle be addressed by better tokenization (see § 8.4), other morphological processes involve more complex transformations of subword units.

Names and technical terms can be handled in a postprocessing step: after first identifying alignments between unknown words in the source and target, we can look up each aligned source word in a dictionary, and choose a replacement (Luong et al., 2015). If the word does not appear in the dictionary, it is likely to be a proper noun, and can be copied directly from the source to the target. This approach can also be integrated directly into the translation model, rather than applying it as a postprocessing step (Jean et al., 2015).

Words with complex internal structure can be handled by translating subword units rather than entire words. A popular technique for identifying subword units is **byte-pair encoding** (BPE; Gage, 1994; Sennrich et al., 2016). The initial vocabulary is defined as the set of characters used in the text. The most common character bigram is then merged into a new symbol, and the vocabulary is updated. The merging operation is applied repeatedly, until the vocabulary reaches some maximum size. For example, given the dictionary  $\{fish, fished, want, wanted, bike, biked\}$ , we would first merge  $e+d$  into the subword unit  $ed$ , since this bigram appears in three words of the six words. Next, there are several bigrams that each appear in a pair of words:  $f+i$ ,  $i+s$ ,  $s+h$ ,  $w+a$ ,  $a+n$ , etc. These can be merged in any order, resulting in the segmentation,  $\{fish, fish+ed, want, want+ed, bik+e, bik+ed\}$ . At this point, there are no subword bigrams that appear more than once. In real data, merging is performed until the number of subword units reaches some predefined threshold,

9805 such as  $10^4$ .

9806 Each subword unit is treated as a token for translation, in both the encoder (source  
 9807 side) and decoder (target side). BPE can be applied jointly to the union of the source and  
 9808 target vocabularies, identifying subword units that appear in both languages. For lan-  
 9809 guages that have different scripts, such as English and Russian, **transliteration** between  
 9810 the scripts should be applied first.<sup>8</sup>

## 9811 18.4 Decoding

Given a trained translation model, the decoding task is:

$$\hat{\mathbf{w}}^{(t)} = \underset{\mathbf{w} \in \mathcal{V}^*}{\operatorname{argmax}} \Psi(\mathbf{w}, \mathbf{w}^{(s)}), \quad [18.45]$$

9812 where  $\mathbf{w}^{(t)}$  is a sequence of tokens from the target vocabulary  $\mathcal{V}$ . It is not possible to  
 9813 efficiently obtain exact solutions to the decoding problem, for even minimally effective  
 9814 models in either statistical or neural machine translation. Today's state-of-the-art transla-  
 9815 tion systems use **beam search** (see § 11.3.1.4), which is an incremental decoding algorithm  
 9816 that maintains a small constant number of competitive hypotheses. Such greedy approxi-  
 9817 mations are reasonably effective in practice, and this may be in part because the decoding  
 9818 objective is only loosely correlated with measures of translation quality, so that exact op-  
 9819 timization of [18.45] may not greatly improve the resulting translations.

Decoding in neural machine translation is somewhat simpler than in phrase-based  
 statistical machine translation.<sup>9</sup> The scoring function  $\Psi$  is defined,

$$\Psi(\mathbf{w}^{(t)}, \mathbf{w}^{(s)}) = \sum_{m=1}^{M^{(t)}} \psi(w_m^{(t)}; \mathbf{w}_{1:m-1}^{(t)}, \mathbf{z}) \quad [18.46]$$

$$\psi(w^{(t)}; \mathbf{w}_{1:m-1}^{(t)}, \mathbf{z}) = \beta_{w_m^{(t)}} \cdot \mathbf{h}_m^{(t)} - \log \sum_{w \in \mathcal{V}} \exp(\beta_w \cdot \mathbf{h}_m^{(t)}), \quad [18.47]$$

9820 where  $\mathbf{z}$  is the encoding of the source sentence  $\mathbf{w}^{(s)}$ , and  $\mathbf{h}_m^{(t)}$  is a function of the encoding  
 9821  $\mathbf{z}$  and the decoding history  $\mathbf{w}_{1:m-1}^{(t)}$ . This formulation subsumes the attentional translation  
 9822 model, where  $\mathbf{z}$  is a matrix encoding of the source.

Now consider the incremental decoding algorithm,

$$\hat{w}_m^{(t)} = \underset{w \in \mathcal{V}}{\operatorname{argmax}} \psi(w; \hat{\mathbf{w}}_{1:m-1}^{(t)}, \mathbf{z}), \quad m = 1, 2, \dots \quad [18.48]$$

---

<sup>8</sup>Transliteration is crucial for converting names and other foreign words between languages that do not share a single script, such as English and Japanese. It is typically approached using the finite-state methods discussed in chapter 9 (Knight and Graehl, 1998).

<sup>9</sup>For more on decoding in phrase-based statistical models, see Koehn (2009).

9823 This algorithm selects the best target language word at position  $m$ , assuming that it has  
 9824 already generated the sequence  $\hat{w}_{1:m-1}^{(t)}$ . (Termination can be handled by augmenting  
 9825 the vocabulary  $\mathcal{V}$  with a special end-of-sequence token, ■.) The incremental algorithm  
 9826 is likely to produce a suboptimal solution to the optimization problem defined in Equa-  
 9827 tion 18.45, because selecting the highest-scoring word at position  $m$  can set the decoder  
 9828 on a “garden path,” in which there are no good choices at some later position  $n > m$ . We  
 9829 might hope for some dynamic programming solution, as in sequence labeling (§ 7.3). But  
 9830 the Viterbi algorithm and its relatives rely on a Markov decomposition of the objective  
 9831 function into a sum of local scores: for example, scores can consider locally adjacent tags  
 9832 ( $y_m, y_{m-1}$ ), but not the entire tagging history  $y_{1:m}$ . This decomposition is not applicable  
 9833 to recurrent neural networks, because the hidden state  $h_m^{(t)}$  is impacted by the entire his-  
 9834 tory  $w_{1:m}^{(t)}$ ; this sensitivity to long-range context is precisely what makes recurrent neural  
 9835 networks so effective.<sup>10</sup> In fact, it can be shown that decoding from any recurrent neural  
 9836 network is NP-complete (Siegelmann and Sontag, 1995; Chen et al., 2018).

9837 **Beam search** Beam search is a general technique for avoiding search errors when ex-  
 9838 haustive search is impossible; it was first discussed in § 11.3.1.4. Beam search can be  
 9839 seen as a variant of the incremental decoding algorithm sketched in Equation 18.48, but  
 9840 at each step  $m$ , a set of  $K$  different hypotheses are kept on the beam. For each hypothesis  
 9841  $k \in \{1, 2, \dots, K\}$ , we compute both the current score  $\sum_{m=1}^{M^{(t)}} \psi(w_{k,m}^{(t)}; w_{k,1:m-1}^{(t)}, z)$  as well as  
 9842 the current hidden state  $h_k^{(t)}$ . At each step in the beam search, the  $K$  top-scoring children  
 9843 of each hypothesis currently on the beam are “expanded”, and the beam is updated. For  
 9844 a detailed description of beam search for RNN decoding, see Graves (2012).

9845 **Learning and search** Conventionally, the learning algorithm is trained to predict the  
 9846 right token in the translation, conditioned on the translation history being correct. But  
 9847 if decoding must be approximate, then we might do better by modifying the learning  
 9848 algorithm to be robust to errors in the translation history. **Scheduled sampling** does this  
 9849 by training on histories that sometimes come from the ground truth, and sometimes come  
 9850 from the model’s own output (Bengio et al., 2015).<sup>11</sup> As training proceeds, the training  
 9851 wheels come off: we increase the fraction of tokens that come from the model rather than  
 9852 the ground truth. Another approach is to train on an objective that relates directly to beam  
 9853 search performance (Wiseman et al., 2016). **Reinforcement learning** has also been applied  
 9854 to decoding of RNN-based translation models, making it possible to directly optimize  
 9855 translation metrics such as BLEU (Ranzato et al., 2016).

---

<sup>10</sup>Note 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.

<sup>11</sup>Scheduled 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.

9856 **18.5 Training towards the evaluation metric**

9857 In likelihood-based training, the objective is to maximize the probability of a parallel  
 9858 corpus. However, translations are not evaluated in terms of likelihood: metrics like BLEU  
 9859 consider only the correctness of a single output translation, and not the range of prob-  
 9860 abilities that the model assigns. It might therefore be better to train translation models  
 9861 to achieve the highest BLEU score possible — to the extent that we believe BLEU mea-  
 9862 sures translation quality. Unfortunately, BLEU and related metrics are not friendly for  
 9863 optimization: they are discontinuous, non-differentiable functions of the parameters of  
 9864 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  $\theta$  that minimize the error of the system's preferred translation,

$$\hat{\mathbf{w}}^{(t)} = \operatorname{argmax}_{\mathbf{w}^{(t)}} \Psi(\mathbf{w}^{(t)}, \mathbf{w}^{(s)}; \theta) \quad [18.49]$$

$$\hat{\theta} = \operatorname{argmin}_{\theta} \Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(s)}) \quad [18.50]$$

9865 However, identifying the top-scoring translation  $\hat{\mathbf{w}}^{(t)}$  is usually intractable, as described  
 9866 in the previous section. In **minimum error-rate training (MERT)**,  $\hat{\mathbf{w}}^{(t)}$  is selected from a  
 9867 set of candidate translations  $\mathcal{Y}(\mathbf{w}^{(s)})$ ; this is typically a strict subset of all possible transla-  
 9868 tions, so that it is only possible to optimize an approximation to the true error rate (Och  
 9869 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  $\theta$  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,

$$R(\theta) = E_{\hat{\mathbf{w}}^{(t)} | \mathbf{w}^{(s)}; \theta} [\Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)})] \quad [18.51]$$

$$= \sum_{\hat{\mathbf{w}}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})} p(\hat{\mathbf{w}}^{(t)} | \mathbf{w}^{(s)}) \times \Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)}). \quad [18.52]$$

9870 **Minimum risk training** minimizes the sum of  $R(\theta)$  across all instances in the training set.

The risk can be generalized by exponentiating the translation probabilities,

$$\tilde{p}(\mathbf{w}^{(t)}; \theta, \alpha) \propto \left( p(\mathbf{w}^{(t)} | \mathbf{w}^{(s)}; \theta) \right)^\alpha \quad [18.53]$$

$$\tilde{R}(\theta) = \sum_{\hat{\mathbf{w}}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})} \tilde{p}(\hat{\mathbf{w}}^{(t)} | \mathbf{w}^{(s)}; \alpha, \theta) \times \Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)}) \quad [18.54]$$

where  $\mathcal{Y}(\mathbf{w}^{(s)})$  is now the set of *all* possible translations for  $\mathbf{w}^{(s)}$ . Exponentiating the probabilities in this way is known as **annealing** (Smith and Eisner, 2006). When  $\alpha = 1$ , then  $\tilde{R}(\boldsymbol{\theta}) = R(\boldsymbol{\theta})$ ; when  $\alpha = \infty$ , then  $\tilde{R}(\boldsymbol{\theta})$  is equivalent to the sum of the errors of the maximum probability translations for each sentence in the dataset.

Clearly the set of candidate translations  $\mathcal{Y}(\mathbf{w}^{(s)})$  is too large to explicitly sum over. Because the error function  $\Delta$  generally does not decompose into smaller parts, there is no efficient dynamic programming solution to sum over this set. We can approximate the sum  $\sum_{\mathbf{w}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})}$  with a sum over a finite number of samples,  $\{\mathbf{w}_1^{(t)}, \mathbf{w}_2^{(t)}, \dots, \mathbf{w}_K^{(t)}\}$ . If these samples were drawn uniformly at random, then the (annealed) risk would be approximated as (Shen et al., 2016),

$$\tilde{R}(\boldsymbol{\theta}) \approx \frac{1}{Z} \sum_{k=1}^K \tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)}; \boldsymbol{\theta}, \alpha) \times \Delta(\mathbf{w}_k^{(t)}, \mathbf{w}^{(t)}) \quad [18.55]$$

$$Z = \sum_{k=1}^K \tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)}; \boldsymbol{\theta}, \alpha). \quad [18.56]$$

Shen et al. (2016) report that performance plateaus at  $K = 100$  for minimum risk training of neural machine translation.

Uniform sampling over the set of all possible translations is undesirable, because most translations have very low probability. A solution from Monte Carlo estimation is **importance sampling**, in which we draw samples from a **proposal distribution**  $q(\mathbf{w}^{(t)})$ . This distribution can be set equal to the current translation model  $p(\mathbf{w}^{(t)} | \mathbf{w}^{(s)}; \boldsymbol{\theta})$ . Each sample is then weighted by an **importance score**,  $\omega_k = \frac{\tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)})}{q(\mathbf{w}_k^{(t)})}$ . The effect of this weighting is to correct for any mismatch between the proposal distribution  $q$  and the true distribution  $\tilde{p}$ . The risk can then be approximated as,

$$\mathbf{w}_k^{(t)} \sim q(\mathbf{w}^{(t)}) \quad [18.57]$$

$$\omega_k = \frac{\tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)})}{q(\mathbf{w}_k^{(t)})} \quad [18.58]$$

$$\tilde{R}(\boldsymbol{\theta}) \approx \frac{1}{\sum_{k=1}^K \omega_k} \sum_{k=1}^K \omega_k \times \Delta(\mathbf{w}_k^{(t)}, \mathbf{w}^{(t)}). \quad [18.59]$$

Importance sampling will generally give a more accurate approximation with a given number of samples. The only formal requirement is that the proposal assigns non-zero probability to every  $\mathbf{w}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})$ . For more on importance sampling and related methods, see Robert and Casella (2013).

**9881 Additional resources**

9882 A complete textbook on machine translation is available from Koehn (2009). While this  
9883 book precedes recent work on neural translation, a more recent draft chapter on neural  
9884 translation models is also available (Koehn, 2017). Neubig (2017) provides a comprehen-  
9885 sive tutorial on neural machine translation, starting from first principles. The course notes  
9886 from Cho (2015) are also useful.

9887 Several neural machine translation systems are available, in connection with each of  
9888 the major neural computing libraries: `lamtram` is an implementation of neural machine  
9889 translation in the `dynet` (Neubig et al., 2017); `OpenNMT` (Klein et al., 2017) is an imple-  
9890 mentation primarily in `Torch`; `tensor2tensor` is an implementation of several of the  
9891 Google translation models in `tensorflow` (Abadi et al., 2016).

9892 Literary translation is especially challenging, even for expert human translators. Mes-  
9893 sud (2014) describes some of these issues in her review of an English translation of *L'étranger*,  
9894 the 1942 French novel by Albert Camus.<sup>12</sup> She compares the new translation by Sandra  
9895 Smith against earlier translations by Stuart Gilbert and Matthew Ward, focusing on the  
9896 difficulties presented by a single word in the first sentence:

9897 Then, too, Smith has reconsidered the book's famous opening. Camus's  
9898 original is deceptively simple: "*Aujourd'hui, maman est morte.*" Gilbert influ-  
9899 enced generations by offering us "Mother died today"—inscribing in Meur-  
9900 sault [the narrator] from the outset a formality that could be construed as  
9901 heartlessness. But *maman*, after all, is intimate and affectionate, a child's name  
9902 for his mother. Matthew Ward concluded that it was essentially untranslatable  
9903 ("mom" or "mummy" being not quite apt), and left it in the original French:  
9904 "Maman died today." There is a clear logic in this choice; but as Smith has  
9905 explained, in an interview in *The Guardian*, *maman* "didn't really tell the reader  
9906 anything about the connotation." She, instead, has translated the sentence as  
9907 "My mother died today."

9908 I chose "My mother" because I thought about how someone would  
9909 tell another person that his mother had died. Meursault is speaking  
9910 to the reader directly. "My mother died today" seemed to me the  
9911 way it would work, and also implied the closeness of "maman" you  
9912 get in the French.

9913 Elsewhere in the book, she has translated *maman* as "mama" — again, striving  
9914 to come as close as possible to an actual, colloquial word that will carry the  
9915 same connotations as *maman* does in French.

---

<sup>12</sup>The book review is currently available online at <http://www.nybooks.com/articles/2014/06/05/camus-new-letranger/>.

9916     The passage is a useful reminder that while the quality of machine translation has  
9917     improved dramatically in recent years, expert human translations draw on considerations  
9918     that are beyond the ken of any known computational approach.

9919   **Exercises**

9920     1. Give a synchronized derivation (§ 18.2.4) for the Spanish-English translation,

9921       (18.4) *El pez enojado atacado.*  
                 The fish angry attacked.  
9922           The angry fish attacked.

9923     As above, the second line shows a word-for-word gloss, and the third line shows  
9924     the desired translation. Use the synchronized production rule in [18.22], and design  
9925     the other production rules necessary to derive this sentence pair. You may derive  
9926     (*atacado*, *attacked*) directly from VP.

# 9927 Chapter 19

## 9928 Text generation

9929 In many of the most interesting problems in natural language processing, language is  
9930 the output. The previous chapter described the specific case of machine translation, but  
9931 there are many other applications, from summarization of research articles, to automated  
9932 journalism, to dialogue systems. This chapter emphasizes three main scenarios: data-to-  
9933 text, in which text is generated to explain or describe a structured record or unstructured  
9934 perceptual input; text-to-text, which typically involves fusing information from multiple  
9935 linguistic sources into a single coherent summary; and dialogue, in which text is generated  
9936 as part of an interactive conversation with one or more human participants.

### 9937 19.1 Data-to-text generation

9938 In data-to-text generation, the input ranges from structured records, such as the descrip-  
9939 tion of an weather forecast (as shown in Figure 19.1), to unstructured perceptual data,  
9940 such as a raw image or video; the output may be a single sentence, such as an image cap-  
9941 tion, or a multi-paragraph argument. Despite this diversity of conditions, all data-to-text  
9942 systems share some of the same challenges (Reiter and Dale, 2000):

- 9943 • determining what parts of the data to describe;
- 9944 • planning a presentation of this information;
- 9945 • **lexicalizing** the data into words and phrases;
- 9946 • organizing words and phrases into well-formed sentences and paragraphs.

9947 The earlier stages of this process are sometimes called **content selection** and **text plan-**  
9948 **ning**; the later stages are often called **surface realization**.

9949 Early systems for data-to-text generation were modular, with separate software com-  
9950 ponents for each task. Artificial intelligence **planning** algorithms can be applied to both

Database:	Temperature			Cloud Sky Cover		
	time	min	mean	max	time	percent (%)
	06:00-21:00	9	15	21	06:00-09:00	25-50
	09:00-12:00				09:00-12:00	50-75
Wind Speed			Wind Direction			
			time	mode		
	06:00-21:00	15	20	30	06:00-21:00	S

Text: Cloudy, with temperatures between 10 and 20 degrees. South wind around 20 mph.

Figure 19.1: An example input-output pair for the task of generating text descriptions of weather forecasts (Konstas and Lapata, 2013). [todo: permission]

the high-level information structure and the organization of individual sentences, ensuring that communicative goals are met (McKeown, 1992; Moore and Paris, 1993). Surface realization can be performed by grammars or templates, which link specific types of data to candidate words and phrases. A simple example template is offered by Wiseman et al. (2017), for generating descriptions of basketball games:

- (19.1) The <team1> (<wins1>-<losses1>) defeated the <team2> (<wins2>-<losses2>),  
<pts1>-<pts2>. The New York Knicks (45-5) defeated the Boston Celtics (11-38), 115-79.

For more complex cases, it may be necessary to apply morphological inflections such as pluralization and tense marking — even in the simple example above, languages such as Russian would require case marking suffixes for the team names. Such inflections can be applied as a postprocessing step. Another difficult challenge for surface realization is the generation of varied **referring expressions** (e.g., *The Knicks*, *New York*, *they*), which is critical to avoid repetition. As discussed in § 16.2.1, the form of referring expressions is constrained by the discourse and information structure.

An example at the intersection of rule-based and statistical techniques is the Nitrogen system (Langkilde and Knight, 1998). The input to Nitrogen is an abstract meaning representation (AMR; see § 13.3) of semantic content to be expressed in a single sentence. In data-to-text scenarios, the abstract meaning representation is the output of a higher-level text planning stage. A set of rules then converts the abstract meaning representation into various sentence plans, which may differ in both the high-level structure (e.g., active versus passive voice) as well as the low-level details (e.g., word and phrase choice). Some examples are shown in Figure 19.2. To control the combinatorial explosion in the number of possible realizations for any given meaning, the sentence plans are unified into a single finite-state acceptor, in which word tokens are represented by arcs (see § 9.1.1). A bigram

```
(a / admire-01
  :ARG0 (v / visitor
    :ARG1-of (c / arrive-01
      :ARG4 (j / Japan)))
  :ARG1 (m / "Mount Fuji"))
```

- Visitors who came to Japan admire Mount Fuji.
- Visitors who came in Japan admire Mount Fuji.
- Mount Fuji is admired by the visitor who came in Japan.

Figure 19.2: Abstract meaning representation and candidate surface realizations from the Nitrogen system. Example adapted from Langkilde and Knight (1998).

language model is then used to compute weights on the arcs, so that the shortest path is also the surface realization with the highest bigram language model probability.

More recent systems are unified models that are trained end-to-end using backpropagation. Data-to-text generation shares many properties with machine translation, including a problem of **alignment**: labeled examples provide the data and the text, but they do not specify which parts of the text correspond to which parts of the data. For example, to learn from Figure 19.1, the system must align the word *cloudy* to records in CLOUD SKY COVER, the phrases *10* and *20 degrees* to the MIN and MAX fields in TEMPERATURE, and so on. As in machine translation, both latent variables and neural attention have been proposed as solutions.

### 19.1.1 Latent data-to-text alignment

Given a dataset of texts and associated records  $\{(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$ , our goal is to learn a model  $\Psi$ , so that

$$\hat{\mathbf{w}} = \underset{\mathbf{w} \in \mathcal{V}^*}{\operatorname{argmax}} \Psi(\mathbf{w}, \mathbf{y}; \theta), \quad [19.1]$$

where  $\mathcal{V}^*$  is the set of strings over a discrete vocabulary, and  $\theta$  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  $\Psi$  into subcomponents. This would be possible if given an **alignment**, specifying which element of  $\mathbf{y}$  is expressed in each part of  $\mathbf{w}$  (Angeli et al., 2010):

$$\Psi(\mathbf{w}, \mathbf{y}; \theta) = \sum_{m=1}^M \psi_{w,y}(\mathbf{w}_m, \mathbf{y}_{z_m}) + \psi_z(z_m, z_{m-1}), \quad [19.2]$$

where  $z_m$  indicates the record aligned to word  $m$ . For example, in Figure 19.1,  $z_1$  might specify that the word *cloudy* is aligned to the record *cloud-sky-cover:percent*. The score for this alignment would then be given by the weight on features such as

$$(\textit{cloudy}, \textit{cloud-sky-cover:percent}), \quad [19.3]$$

which could be learned from labeled data  $\{(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})\}_{i=1}^N$ . The function  $\psi_z$  can learn to assign higher scores to alignments that are coherent, referring to the same records in adjacent parts of the text.<sup>1</sup>

Several datasets include structured records and natural language text (Barzilay and McKeown, 2005; Chen and Mooney, 2008; Liang and Klein, 2009), but the alignments between text and records are usually not available.<sup>2</sup> One solution is to model the problem probabilistically, treating the alignment as a latent variable (Liang et al., 2009; Konstas and Lapata, 2013). The model can then be estimated using expectation maximization or sampling (see chapter 5).

### 19.1.2 Neural data-to-text generation

The **encoder-decoder model** and **neural attention** were introduced in § 18.3 as methods for neural machine translation. They can also be applied to data-to-text generation, with the data acting as the source language (Mei et al., 2016). In neural machine translation, the attention mechanism linked words in the source to words in the target; in data-to-text generation, the attention mechanism can link each part of the generated text back to a record in the data. The biggest departure from translation is in the encoder, which depends on the form of the data.

#### 19.1.2.1 Data encoders

In some types of structured records, all values are drawn from discrete sets. For example, the birthplace of an individual is drawn from a discrete set of possible locations; the diagnosis and treatment of a patient are drawn from an exhaustive list of clinical codes (Johnson et al., 2016). In such cases, vector embeddings can be estimated for each field and possible value: for example, a vector embedding for the field BIRTHPLACE, and another embedding for the value BERKELEY\_CALIFORNIA (Bordes et al., 2011). The table of such embeddings serves as the encoding of a structured record (He et al., 2017). It is also possible to compress the entire table into a single vector representation, by **pooling** across the embeddings of each field and value (Lebret et al., 2016).

**Sequences** Some types of structured records have a natural ordering, such as events in a game (Chen and Mooney, 2008) and steps in a recipe (Tutin and Kittredge, 1992). For example, the following records describe a sequence of events in a robot soccer match (Mei

---

<sup>1</sup>More expressive decompositions of  $\Psi$  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.

<sup>2</sup>An exception is a dataset of records and summaries from American football games, containing annotations of alignments between sentences and records (Snyder and Barzilay, 2007).

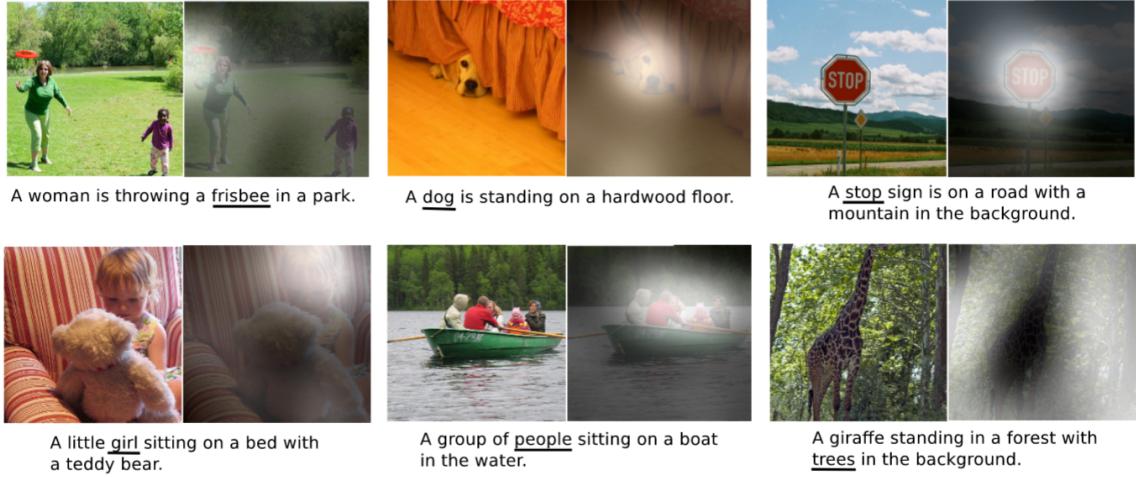


Figure 19.3: Examples of the image captioning task, with attention masks shown for each of the underlined words. From Xu et al. (2015). [todo: permission]

et al., 2016):

```
PASS(arg1 = PURPLE6,arg2 = PURPLE3)
KICK(arg1 = PURPLE3)
BADPASS(arg1 = PURPLE3,arg2 = PINK9).
```

10025 Each event is a single record, and can be encoded by a concatenation of vector representations for the event type (e.g., PASS), the field (e.g., arg1), and the values (e.g., PURPLE3),  
10026 e.g.,  
10027

$$\mathbf{X} = [\mathbf{u}_{\text{PASS}}, \mathbf{u}_{\text{arg1}}, \mathbf{u}_{\text{PURPLE6}}, \mathbf{u}_{\text{arg2}}, \mathbf{u}_{\text{PURPLE3}}]. \quad [19.4]$$

10028 This encoding can then act as the input layer for a recurrent neural network, yielding a  
10029 sequence of vector representations  $\{z_r\}_{r=1}^R$ , where  $r$  indexes over records. Interestingly,  
10030 this sequence-based approach is effective even in cases where there is no natural ordering  
10031 over the records, such as the weather data in Figure 19.1 (Mei et al., 2016).

10032 **Images** Another flavor of data-to-text generation is the generation of text captions for  
10033 images. Examples from this task are shown in Figure 19.3. Images are naturally represented as tensors: a color image of  $320 \times 240$  pixels would be stored as a tensor with  
10034  $320 \times 240 \times 3$  intensity values. The dominant approach to image classification is to encode images as vectors using a combination of convolution and pooling (Krizhevsky et al.,  
10035 2012). Chapter 3 explains how to use convolutional networks for text; for images, convolution is applied across the vertical, horizontal, and color dimensions. By pooling the results of successive convolutions, the image is converted to a vector representation, which

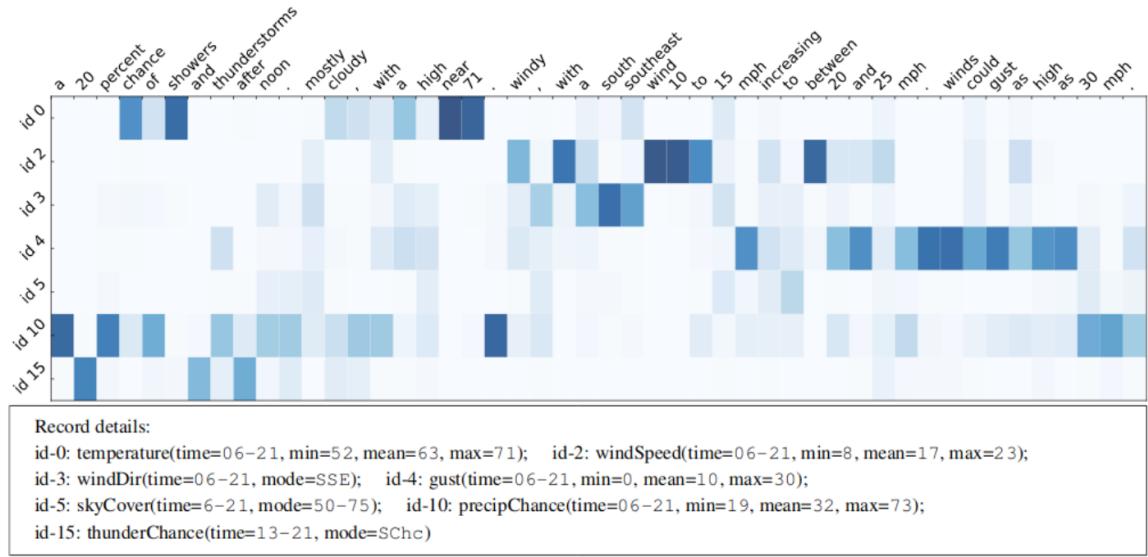


Figure 19.4: Neural attention in text generation. Figure from Mei et al. (2016).[todo: permission]

10040 can then be fed directly into the decoder as the initial state (Vinyals et al., 2015), just as  
 10041 in the sequence-to-sequence translation model (see § 18.3). Alternatively, one can apply  
 10042 a set of convolutional networks, yielding vector representations for different parts of the  
 10043 image, which can then be combined using neural attention (Xu et al., 2015).

#### 10044 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  $\mathbf{c}_m$ ,

$$\psi_\alpha(m, r) = \beta_\alpha \cdot f(\Theta_\alpha[\mathbf{h}_m; \mathbf{z}_r]) \quad [19.5]$$

$$\boldsymbol{\alpha}_m = \text{SoftMax}([\psi_\alpha(m, 1), \psi_\alpha(m, 2), \dots, \psi_\alpha(m, R)]) \quad [19.6]$$

$$\mathbf{c}_m = \sum_{r=1}^R \alpha_{m \rightarrow r} \mathbf{z}_r, \quad [19.7]$$

10045 where  $f$  is an elementwise nonlinearity such as tanh or ReLU (see § 3.2.1). The weighted  
 10046 average  $\mathbf{c}_m$  can then be included in the recurrent update to the decoder state, or in the  
 10047 emission probabilities, as described in § 18.3.1. Figure 19.4 shows the attention to compo-  
 10048 nents of a weather record, while generating the text shown on the  $x$ -axis.

10049 Adapting this architecture to image captioning is straightforward. A convolutional  
 10050 neural networks is applied to a set of image locations, and the output at each location  $\ell$  is  
 10051 represented with a vector  $z_\ell$ . Attention can then be computed over the image locations,  
 10052 as shown in the right panels of each pair of images in Figure 19.3.

10053 Various modifications to this basic mechanism have been proposed. In **coarse-to-fine**  
 10054 **attention** (Mei et al., 2016), each record receives a global attention  $a_r \in [0, 1]$ , which is in-  
 10055 dependent of the decoder state. This global attention, which represents the overall impor-  
 10056 tance of the record, is multiplied with the decoder-based attention scores, before comput-  
 10057 ing the final normalized attentions. In **structured attention**, the attention vector  $\alpha_{m \rightarrow \cdot}$  can  
 10058 include structural biases, which can favor assigning higher attention values to contiguous  
 10059 segments or to dependency subtrees (Kim et al., 2017). Structured attention vectors can  
 10060 be computed by running the forward-backward algorithm to obtain marginal attention  
 10061 probabilities (see § 7.5.3.3). Because each step in the forward-backward algorithm is dif-  
 10062 ferentiable, it can be encoded in a computation graph, and end-to-end learning can be  
 10063 performed by backpropagation.

### 10064 19.1.2.3 Decoder

10065 Given the encoding, the decoder can function just as in neural machine translation (see  
 10066 § 18.3.1), using the attention-weighted encoder representation in the decoder recurrence  
 10067 and/or output computation. As in machine translation, beam search can help to avoid  
 10068 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).<sup>3</sup> First introduce an additional variable  $s_m \in \{\text{gen}, \text{copy}\}$ , indicating whether token  $w_m^{(t)}$  should be generated or copied. The decoder probability is then,

$$p(w^{(t)} | w_{1:m-1}^{(t)}, \mathbf{Z}, s_m) = \begin{cases} \text{SoftMax}(\beta_{w^{(t)}} \cdot h_{m-1}^{(t)}), & s_m = \text{gen} \\ \sum_{r=1}^R \delta(w_r^{(s)} = w^{(t)}) \times \alpha_{m \rightarrow r}, & s_m = \text{copy}, \end{cases} \quad [19.8]$$

10069 where  $\delta(w_r^{(s)} = w^{(t)})$  is an indicator function, taking the value 1 iff the text of the record  
 10070  $w_r^{(s)}$  is identical to the target word  $w^{(t)}$ . The probability of copying record  $r$  from the source  
 10071 is  $\delta(s_m = \text{copy}) \times \alpha_{m \rightarrow r}$ , the product of the copy probability by the local attention. Note  
 10072 that in this model, the attention weights  $\alpha_m$  are computed from the *previous* decoder state  
 10073  $h_{m-1}$ . The computation graph therefore remains a feedforward network, with recurrent  
 10074 paths such as  $h_{m-1}^{(t)} \rightarrow \alpha_m \rightarrow w_m^{(t)} \rightarrow h_m^{(t)}$ .

---

<sup>3</sup>A 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.

10075 To facilitate end-to-end training, the switching variable  $s_m$  can be represented by a  
 10076 gate  $\pi_m$ , which is computed from a two-layer feedforward network, whose input consists  
 10077 of the concatenation of the decoder state  $\mathbf{h}_{m-1}^{(t)}$  and the attention-weighted representation  
 10078 of the data,  $\mathbf{c}_m = \sum_{r=1}^R \alpha_{m \rightarrow r} \mathbf{z}_r$ ,

$$\pi_m = \sigma(\Theta^{(2)} f(\Theta^{(1)}[\mathbf{h}_{m-1}^{(t)}; \mathbf{c}_m])). \quad [19.9]$$

The full generative probability at token  $m$  is then,

$$p(w^{(t)} | \mathbf{w}_{1:m}^{(t)}, \mathbf{Z}) = \pi_m \times \underbrace{\frac{\exp \beta_{w^{(t)}} \cdot \mathbf{h}_{m-1}^{(t)}}{\sum_{j=1}^V \exp \beta_j \cdot \mathbf{h}_{m-1}^{(t)}}}_{\text{generate}} + (1 - \pi_m) \times \underbrace{\sum_{r=1}^R \delta(w_r^{(s)} = w^{(t)}) \times \alpha_{m \rightarrow r}}_{\text{copy}}. \quad [19.10]$$

## 10079 19.2 Text-to-text generation

10080 Text-to-text generation includes problems of summarization and simplification:

- 10081 • reading a novel and outputting a paragraph-long summary of the plot;<sup>4</sup>
- 10082 • reading a set of blog posts about politics, and outputting a bullet list of the various  
10083 issues and perspectives;
- 10084 • reading a technical research article about the long-term health consequences of drink-  
10085 ing kombucha, and outputting a summary of the article in language that non-experts  
10086 can understand.

10087 These problems can be approached in two ways: through the encoder-decoder architec-  
10088 ture discussed in the previous section, or by operating directly on the input text.

### 10089 19.2.1 Neural abstractive summarization

10090 **Sentence summarization** is the task of shortening a sentence while preserving its mean-  
10091 ing, as in the following examples (Knight and Marcu, 2000; Rush et al., 2015):

- 10092 (19.2) The documentation is typical of Epson quality: excellent.  
10093 Documentation is excellent.

---

<sup>4</sup>In § 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.

- 10095 (19.3) Russian defense minister Ivanov called sunday for the creation of a joint front for  
 10096 combating global terrorism.  
 10097 Russia calls for joint front against terrorism.  
 10098

10099 Sentence summarization is closely related to **sentence compression**, in which the sum-  
 10100 mary is produced by deleting words or phrases from the original (Clarke and Lapata,  
 10101 2008). But as shown in (19.3), a sentence summary can also introduce new words, such as  
 10102 *against*, which replaces the phrase *for combatting*.

10103 Sentence summarization can be treated as a machine translation problem, using the at-  
 10104 tentional encoder-decoder translation model discussed in § 18.3.1 (Rush et al., 2015). The  
 10105 longer sentence is encoded into a sequence of vectors, one for each token. The decoder  
 10106 then computes attention over these vectors when updating its own recurrent state. As  
 10107 with data-to-text generation, it can be useful to augment the encoder-decoder model with  
 10108 the ability to copy words directly from the source. Rush et al. (2015) train this model by  
 10109 building four million sentence pairs from news articles. In each pair, the longer sentence is  
 10110 the first sentence of the article, and the summary is the article headline. Sentence summa-  
 10111 rization can also be trained in a semi-supervised fashion, using a probabilistic formulation  
 10112 of the encoder-decoder model called a **variational autoencoder** (Miao and Blunsom, 2016,  
 10113 also see § 14.8.2).

When summarizing longer documents, an additional concern is that the summary not be repetitive: each part of the summary should cover new ground. This can be addressed by maintaining a vector of the sum total of all attention values thus far,  $t_m = \sum_{n=1}^m \alpha_n$ . This total can be used as an additional input to the computation of the attention weights,

$$\alpha_{m \rightarrow n} \propto \exp \left( \mathbf{v}_\alpha \cdot \tanh(\Theta_\alpha[\mathbf{h}_m^{(t)}; \mathbf{h}_n^{(s)}; \mathbf{t}_m]) \right), \quad [19.11]$$

which enables the model to learn to prefer parts of the source which have not been attended to yet (Tu et al., 2016). To further encourage diversity in the generated summary, See et al. (2017) introduce a **coverage loss** to the objective function,

$$\ell_m = \sum_{n=1}^{M^{(s)}} \min(\alpha_{m \rightarrow n}, t_{m \rightarrow n}). \quad [19.12]$$

10114 This loss will be low if  $\alpha_{m \rightarrow \cdot}$  assigns little attention to words that already have large  
 10115 values in  $t_{m \rightarrow \cdot}$ . Coverage loss is similar to the concept of **marginal relevance**, in which  
 10116 the reward for adding new content is proportional to the extent to which it increases  
 10117 the overall amount of information conveyed by the summary (Carbonell and Goldstein,  
 10118 1998).

10119 **19.2.2 Sentence fusion for multi-document summarization**

10120 In **multi-document summarization**, the goal is to produce a summary that covers the  
 10121 content of several documents (McKeown et al., 2002). One approach to this challenging  
 10122 problem is to identify sentences across multiple documents that relate to a single theme,  
 10123 and then to fuse them into a single sentence (Barzilay and McKeown, 2005). As an exam-  
 10124 ple, consider the following two sentences (McKeown et al., 2010):

- 10125 (19.4) Palin actually turned against the bridge project only after it became a national  
 10126 symbol of wasteful spending.  
 10127 (19.5) Ms. Palin supported the bridge project while running for governor, and aban-  
 10128 doned it after it became a national scandal.

10129 An *intersection* preserves only the content that is present in both sentences:

- 10130 (19.6) Palin turned against the bridge project after it became a national scandal.

10131 A *union* includes information from both sentences:

- 10132 (19.7) Ms. Palin supported the bridge project while running for governor, but turned  
 10133 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),

$$\max_{\mathbf{y}} \sum_{i,j,r} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) \times y_{i,j,r} \quad [19.13]$$

$$\text{s.t. } \mathbf{y} \in \mathcal{C}, \quad [19.14]$$

10134 where the variable  $y_{i,j,r} \in \{0, 1\}$  indicates whether there is an edge from  $i$  to  $j$  of type  $r$ ,  
 10135 the score of this edge is  $\psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})$ , and  $\mathcal{C}$  is a set of constraints, described below. As  
 10136 usual,  $\mathbf{w}$  is the list of words in the graph, and  $\boldsymbol{\theta}$  is a vector of parameters. The score  $\psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})$   
 10137 reflects the “importance” of the modifier  $j$  to the overall meaning: in intersective  
 10138 fusion, this score indicates the extent to which the content in this edge is expressed in all  
 10139 sentences; in union fusion, the score indicates whether the content in the edge is expressed  
 10140 in any sentence.

10141 The constraint set  $\mathcal{C}$  ensures that  $\mathbf{y}$  forms a valid dependency graph. It can also im-  
 10142 pose additional linguistic constraints: for example, ensuring that coordinated nouns are

10143 sufficiently similar. The resulting tree must then be **linearized** into a sentence. This is  
 10144 typically done by generating a set of candidate linearizations, and choosing the one with  
 10145 the highest score under a language model (Langkilde and Knight, 1998; Song et al., 2016).

## 10146 19.3 Dialogue

10147 **Dialogue systems** are capable of conversing with a human interlocutor, often to per-  
 10148 form some task (Grosz, 1979), but sometimes just to chat (Weizenbaum, 1966). While re-  
 10149 search on dialogue systems goes back several decades (Carbonell, 1970; Winograd, 1972),  
 10150 commercial systems such as Alexa and Siri have recently brought this technology into  
 10151 widespread use. Nonetheless, there is a significant gap between research and practice:  
 10152 many practical dialogue systems remain scripted and inflexible, while research systems  
 10153 emphasize abstractive text generation, “on-the-fly” decision making, and probabilistic  
 10154 reasoning about the user’s intentions.

### 10155 19.3.1 Finite-state and agenda-based dialogue systems

10156 Finite-state automata were introduced in chapter 9 as a formal model of computation,  
 10157 in which string inputs and outputs are linked to transitions between a finite number of  
 10158 discrete states. This model naturally fits simple task-oriented dialogues, such as the one  
 10159 shown in the left panel of Figure 19.5. This (somewhat frustrating) dialogue can be repre-  
 10160 sented with a finite-state transducer, as shown in the right panel of the figure. The accept-  
 10161 ing state is reached only when the two needed pieces of information are provided, and the  
 10162 human user confirms that the order is correct. In this simple scenario, the TOPPING and  
 10163 ADDRESS are the two **slots** associated with the activity of ordering a pizza, which is called  
 10164 a **frame**. Frame representations can be hierarchical: for example, an ADDRESS could have  
 10165 slots of its own, such as STREET and CITY.

10166 In the example dialogue in Figure 19.5, the user provides the precise inputs that are  
 10167 needed in each turn (e.g., *anchovies*; *the College of Computing building*). Some users may  
 10168 prefer to communicate more naturally, with phrases like *I’d, uh, like some anchovies please*.  
 10169 One approach to handling such utterances is to design a custom grammar, with non-  
 10170 terminals for slots such as TOPPING and LOCATION. However, context-free parsing of  
 10171 unconstrained speech input is challenging. A more lightweight alternative is BIO-style  
 10172 sequence labeling (see § 8.3), e.g.:

10173 (19.9) *I’d like anchovies , and please bring it to the College of Computing*  
 O O B-TOPPING O O O O O O O B-ADDR I-ADDR I-ADDR I-ADDR I-ADDR  
 10174 *Building .*  
 I-ADDR O

(19.8) A: I want to order a pizza.  
 B: What toppings?  
 A: Anchovies.  
 B: Ok, what address?  
 A: The College of Computing building.  
 B: Please confirm: one pizza with artichokes, to be delivered to the College of Computing building.  
 A: No.  
 B: What toppings?  
 ...

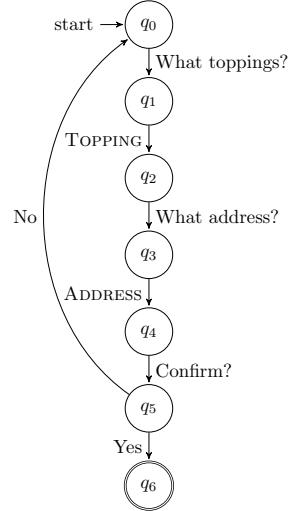


Figure 19.5: An example dialogue and the associated finite-state model. In the finite-state model, SMALL CAPS indicates that the user must provide information of this type in their answer.

10175 The tagger can be driven by a bi-directional recurrent neural network, similar to recurrent  
 10176 approaches to semantic role labeling described in § 13.2.3.

10177 The input in (19.9) could not be handled by the finite-state system from Figure 19.5,  
 10178 which forces the user to provide the topping first, and then the location. In this sense,  
 10179 the **initiative** is driven completely by the system. **Agenda-based dialogue systems** ex-  
 10180 tend finite-state architectures by attempting to recognize all slots that are filled by the  
 10181 user’s reply, thereby handling these more complex examples. Agenda-based systems dy-  
 10182 namically pose additional questions until the frame is complete (Bobrow et al., 1977; Allen  
 10183 et al., 1995; Rudnicky and Xu, 1999). Such systems are said to be **mixed-initiative**, because  
 10184 both the user and the system can drive the direction of the dialogue.

### 10185 19.3.2 Markov decision processes

10186 The task of dynamically selecting the next move in a conversation is known as **dialogue**  
 10187 **management**. This problem can be framed as a **Markov decision process**, which is a  
 10188 theoretical model that includes a discrete set of states, a discrete set of actions, a function  
 10189 that computes the probability of transitions between states, and a function that computes  
 10190 the cost or reward of action-state pairs. Let’s see how each of these elements pertains to  
 10191 the pizza ordering dialogue system.

- 10192 • Each state is a tuple of information about whether the topping and address are

10193 known, and whether the order has been confirmed. For example,

$$(KNOWN\ TOPPING,\ UNKNOWN\ ADDRESS,\ NOT\ CONFIRMED) \quad [19.15]$$

10194 is a possible state. Any state in which the pizza order is confirmed is a terminal  
 10195 state, and the Markov decision process stops after entering such a state.

- 10196 • The set of actions includes querying for the topping, querying for the address, and  
 10197 requesting confirmation. Each action induces a probability distribution over states,  
 10198  $p(s_t | a_t, s_{t-1})$ . For example, requesting confirmation of the order is not likely to  
 10199 result in a transition to the terminal state if the topping is not yet known. This  
 10200 probability distribution over state transitions may be learned from data, or it may  
 10201 be specified in advance.
- 10202 • Each state-action-state tuple earns a reward,  $r_a(s_t, s_{t+1})$ . In the context of the pizza  
 10203 ordering system, a simple reward function would be,

$$r_a(s_t, s_{t+1}) = \begin{cases} 0, & a = \text{CONFIRM}, s_{t+1} = (*, *, \text{CONFIRMED}) \\ -10, & a = \text{CONFIRM}, s_{t+1} = (*, *, \text{NOT CONFIRMED}) \\ -1, & a \neq \text{CONFIRM} \end{cases} \quad [19.16]$$

10204 This function assigns zero reward for successful transitions to the terminal state, a  
 10205 large negative reward to a rejected request for confirmation, and a small negative re-  
 10206 ward for every other type of action. The system is therefore rewarded for reaching  
 10207 the terminal state in few steps, and penalized for prematurely requesting confirma-  
 10208 tion.

10209 In a Markov decision process, a **policy** is a function  $\pi : \mathcal{S} \mapsto \mathcal{A}$  that maps from states to  
 10210 actions (see § 15.2.4.3). The value of a policy is the expected sum of discounted rewards,  
 10211  $E_\pi[\sum_{t=1}^T \gamma^t r_{a_t}(s_t, s_{t+1})]$ , where  $\gamma$  is the discount factor,  $\gamma \in [0, 1)$ . Discounting has the  
 10212 effect of emphasizing rewards that can be obtained immediately over less certain rewards  
 10213 in the distant future.

10214 An optimal policy can be obtained by dynamic programming, by iteratively updating  
 10215 the **value function**  $V(s)$ , which is the expectation of the cumulative reward from  $s$  under  
 10216 the optimal action  $a$ ,

$$V(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s' | s, a)[r_a(s, s') + \gamma V(s')]. \quad [19.17]$$

10217 The value function  $V(s)$  is computed in terms of  $V(s')$  for all states  $s' \in \mathcal{S}$ . A series  
 10218 of iterative updates to the value function will eventually converge to a stationary point.  
 10219 This algorithm is known as **value iteration**. Given the converged value function  $V(s)$ , the

10220 optimal action at each state is the argmax,

$$\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s' | s, a)[r_a(s, s') + \gamma V(s')]. \quad [19.18]$$

10221 Value iteration and related algorithms are described in detail by Sutton and Barto (1998).  
 10222 For applications to dialogue systems, see Levin et al. (1998) and Walker (2000).

10223 The Markov decision process framework assumes that the current state of the dialogue  
 10224 is known. In reality, the system may misinterpret the user’s statements — for example,  
 10225 believing that a specification of the delivery location (PEACHTREE) is in fact a specification  
 10226 of the topping (PEACHES). In a **partially observable Markov decision process (POMDP)**,  
 10227 the system receives an *observation*  $o$ , which is probabilistically conditioned on the state,  
 10228  $p(o | s)$ . It must therefore maintain a distribution of beliefs about which state it is in, with  
 10229  $q_t(s)$  indicating the degree of belief that the dialogue is in state  $s$  at time  $t$ . The POMDP  
 10230 formulation can help to make dialogue systems more robust to errors, particularly in the  
 10231 context of spoken language dialogues, where the speech itself may be misrecognized (Roy  
 10232 et al., 2000; Williams and Young, 2007). However, finding the optimal policy in a POMDP  
 10233 is computationally intractable, requiring additional approximations.

### 10234 19.3.3 Neural chatbots

10235 Chatting is a lot easier when you don’t need to get anything done. **Chatbots** are systems  
 10236 that parry the user’s input with a response that keeps the conversation going. They can be  
 10237 built from the encoder-decoder architecture discussed in § 18.3 and § 19.1.2: the encoder  
 10238 converts the user’s input into a vector, and the decoder produces a sequence of words as a  
 10239 response. For example, Shang et al. (2015) apply the attentional encoder-decoder transla-  
 10240 tion model, training on a dataset of posts and responses from the Chinese microblogging  
 10241 platform Sina Weibo.<sup>5</sup> This approach is capable of generating replies that relate themati-  
 10242 cally to the input, as shown in the following examples:<sup>6</sup>

10243 (19.10) A: High fever attacks me every New Year’s day.  
 10244 Get B: well soon and stay healthy!

10245 (19.11) A: I gain one more year. Grateful to my group, so happy.  
 10246 B: Getting old now. Time has no mercy.

10247 While encoder-decoder models can generate responses that make sense in the con-  
 10248 text of the immediately preceding turn, they struggle to maintain coherence over longer

---

<sup>5</sup>Twitter 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).

<sup>6</sup>All examples are translated from Chinese by Shang et al. (2015).

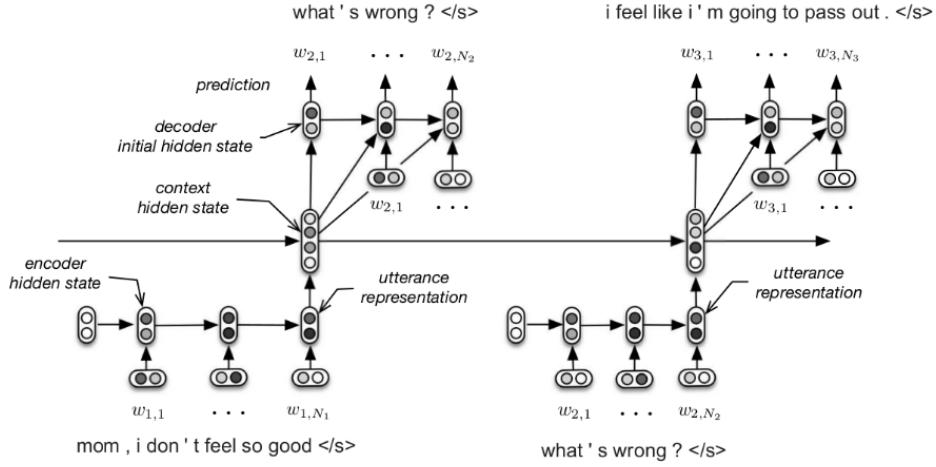


Figure 19.6: A hierarchical recurrent neural network for dialogue, with recurrence over both words and turns, from Serban et al. (2016). [todo: permission]

conversations. One solution is to model the dialogue context recurrently. This creates a **hierarchical recurrent network**, including both word-level and turn-level recurrences. The turn-level hidden state is then used as additional context in the decoder (Serban et al., 2016), as shown in Figure 19.6.

An open question is how to integrate the encoder-decoder architecture into task-oriented dialogue systems. Neural chatbots can be trained end-to-end: the user’s turn is analyzed by the encoder, and the system output is generated by the decoder. This architecture can be trained by log-likelihood using backpropagation (e.g., Sordoni et al., 2015; Serban et al., 2016), or by more elaborate objectives, using reinforcement learning (Li et al., 2016). In contrast, the task-oriented dialogue systems described in § 19.3.1 typically involve a set of specialized modules: one for recognizing the user input, another for deciding what action to take, and a third for arranging the text of the system output.

Recurrent neural network decoders can be integrated into Markov Decision Process dialogue systems, by conditioning the decoder on a representation of the information that is to be expressed in each turn (Wen et al., 2015). Specifically, the long short-term memory (LSTM; § 6.3) architecture is augmented so that the memory cell at turn  $m$  takes an additional input  $d_m$ , which is a representation of the slots and values to be expressed in the next turn. However, this approach still relies on additional modules to recognize the user’s utterance and to plan the overall arc of the dialogue.

Another promising direction is to create embeddings for the elements in the domain: for example, the slots in a record and the entities that can fill them. The encoder then

10270 encodes not only the words of the user’s input, but the embeddings of the elements that  
 10271 the user mentions. Similarly, the decoder is endowed with the ability to refer to specific  
 10272 elements in the knowledge base. He et al. (2017) show that such a method can learn to  
 10273 play a collaborative dialogue game, in which both players are given a list of entities and  
 10274 their properties, and the goal is to find an entity that is on both players’ lists.

## 10275 Further reading

10276 Gatt and Krahmer (2018) provide a comprehensive recent survey on text generation. For  
 10277 a book-length treatment of earlier work, see Reiter and Dale (2000). For a survey on image  
 10278 captioning, see Bernardi et al. (2016); for a survey of pre-neural approaches to dialogue  
 10279 systems, see Rieser and Lemon (2011). **Dialogue acts** were introduced in § 8.6 as a labeling  
 10280 scheme for human-human dialogues; they also play a critical role in task-based dialogue  
 10281 systems (e.g., Allen et al., 1996). The incorporation of theoretical models of dialogue into  
 10282 computational systems is reviewed by Jurafsky and Martin (2009, chapter 24).

10283 While this chapter has focused on the informative dimension of text generation, another  
 10284 line of research aims to generate text with configurable stylistic properties (Walker  
 10285 et al., 1997; Mairesse and Walker, 2011; Ficler and Goldberg, 2017; Hu et al., 2017). This  
 10286 chapter also does not address the generation of creative text such as narratives (Riedl and  
 10287 Young, 2010), jokes (Ritchie, 2001), poems (Colton et al., 2012), and song lyrics (Gonçalo Oliveira  
 10288 et al., 2007).

## 10289 Exercises

10290 1. The SimpleNLG system produces surface realizations from representations of de-  
 10291 sired syntactic structure (Gatt and Reiter, 2009). This system can be accessed on  
 10292 github at <https://github.com/simpleNLG/simpleNLG>. Download the sys-  
 10293 tem, and produce realizations of the following examples:

- 10294 (19.12) Call me Ismael.
- 10295 (19.13) I try all things.
- 10296 (19.14) I achieve what I can.

10297 Then convert each example to a question. [todo: Can’t get SimpleNLG to work with  
 10298 python anymore]

10299 **Appendix A**

10300 **Probability**

10301 Probability theory provides a way to reason about random events. The sorts of random  
10302 events that are typically used to explain probability theory include coin flips, card draws,  
10303 and the weather. It may seem odd to think about the choice of a word as akin to the flip of  
10304 a coin, particularly if you are the type of person to choose words carefully. But random or  
10305 not, language has proven to be extremely difficult to model deterministically. Probability  
10306 offers a powerful tool for modeling and manipulating linguistic data.

10307 Probability can be thought of in terms of **random outcomes**: for example, a single coin  
10308 flip has two possible outcomes, heads or tails. The set of possible outcomes is the **sample**  
10309 **space**, and a subset of the **sample space** is an **event**. For a sequence of two coin flips,  
10310 there are four possible outcomes,  $\{HH, HT, TH, TT\}$ , representing the ordered sequences  
10311 heads-head, heads-tails, tails-heads, and tails-tails. The event of getting exactly one head  
10312 includes two outcomes:  $\{HT, TH\}$ .

10313 Formally, a probability is a function from events to the interval between zero and one:  
10314  $\Pr : \mathcal{F} \mapsto [0, 1]$ , where  $\mathcal{F}$  is the set of possible events. An event that is certain has proba-  
10315 bility one; an event that is impossible has probability zero. For example, the probability  
10316 of getting fewer than three heads on two coin flips is one. Each outcome is also an event  
10317 (a set with exactly one element), and for two flips of a fair coin, the probability of each  
10318 outcome is,

$$\Pr(\{HH\}) = \Pr(\{HT\}) = \Pr(\{TH\}) = \Pr(\{TT\}) = \frac{1}{4}. \quad [\text{A.1}]$$

10319 **A.1 Probabilities of event combinations**

10320 Because events are sets of outcomes, we can use set-theoretic operations such as comple-  
10321 ment, intersection, and union to reason about the probabilities of events and their combi-  
10322 nations.

10323 For any event  $A$ , there is a **complement**  $\neg A$ , such that:

- 10324 • The probability of the union  $A \cup \neg A$  is  $\Pr(A \cup \neg A) = 1$ ;
- 10325 • The intersection  $A \cap \neg A = \emptyset$  is the empty set, and  $\Pr(A \cap \neg A) = 0$ .

10326 In the coin flip example, the event of obtaining a single head on two flips corresponds to  
 10327 the set of outcomes  $\{HT, TH\}$ ; the complement event includes the other two outcomes,  
 10328  $\{TT, HH\}$ .

### 10329 A.1.1 Probabilities of disjoint events

10330 When two events have an empty intersection,  $A \cap B = \emptyset$ , they are **disjoint**. The probabili-  
 10331 ty of the union of two disjoint events is equal to the sum of their probabilities,

$$A \cap B = \emptyset \Rightarrow \Pr(A \cup B) = \Pr(A) + \Pr(B). \quad [A.2]$$

10332 This is the **third axiom of probability**, and it can be generalized to any countable sequence  
 10333 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,

$$\Pr(\{HT, TH\}) = \Pr(\{HT\} \cup \{TH\}) = \Pr(\{HT\}) + \Pr(\{TH\}) \quad [A.3]$$

$$= \frac{1}{4} + \frac{1}{4} = \frac{1}{2}. \quad [A.4]$$

10334 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:

$$\Pr(B - (A \cap B)) = \Pr(B) - \Pr(A \cap B) \quad [A.8]$$

$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B). \quad [A.9]$$

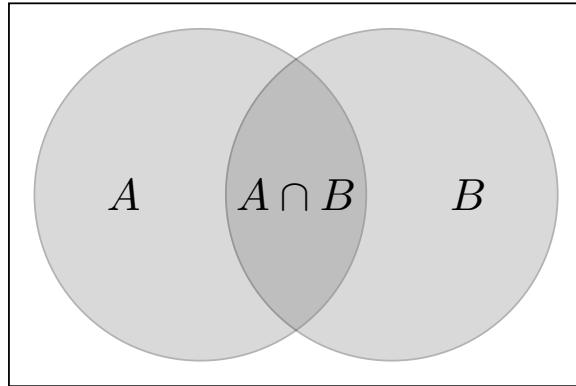


Figure A.1: A visualization of the probability of non-disjoint events  $A$  and  $B$ .

### 10335 A.1.2 Law of total probability

10336 A set of events  $\mathcal{B} = \{B_1, B_2, \dots, B_N\}$  is a **partition** of the sample space iff each pair of  
 10337 events is disjoint ( $B_i \cap B_j = \emptyset$ ), and the union of the events is the entire sample space.  
 10338 The law of total probability states that we can **marginalize** over these events as follows,

$$\Pr(A) = \sum_{B_n \in \mathcal{B}} \Pr(A \cap B_n). \quad [\text{A.10}]$$

10339 For any event  $B$ , the union  $B \cup \neg B$  is a partition of the sample space. Therefore, a special  
 10340 case of the law of total probability is,

$$\Pr(A) = \Pr(A \cap B) + \Pr(A \cap \neg B). \quad [\text{A.11}]$$

## 10341 A.2 Conditional probability and Bayes' rule

A **conditional probability** is an expression like  $\Pr(A \mid B)$ , which is the probability of the event  $A$ , assuming that event  $B$  happens too. For example, we may be interested in the probability of a randomly selected person answering the phone by saying *hello*, conditioned on that person being a speaker of English. Conditional probability is defined as the ratio,

$$\Pr(A \mid B) = \frac{\Pr(A \cap B)}{\Pr(B)}. \quad [\text{A.12}]$$

The **chain rule of probability** states that  $\Pr(A \cap B) = \Pr(A \mid B) \times \Pr(B)$ , which is just

a rearrangement of terms from Equation A.12. The chain rule can be applied repeatedly:

$$\begin{aligned}\Pr(A \cap B \cap C) &= \Pr(A | B \cap C) \times \Pr(B \cap C) \\ &= \Pr(A | B \cap C) \times \Pr(B | C) \times \Pr(C).\end{aligned}$$

**Bayes' rule** (sometimes called Bayes' law or Bayes' theorem) gives us a way to convert between  $\Pr(A | B)$  and  $\Pr(B | A)$ . It follows from the definition of conditional probability and the chain rule:

$$\Pr(A | B) = \frac{\Pr(A \cap B)}{\Pr(B)} = \frac{\Pr(B | A) \times \Pr(A)}{\Pr(B)} \quad [\text{A.13}]$$

10342 Each term in Bayes rule has a name, which we will occasionally use:

- 10343 • Pr( $A$ ) is the **prior**, since it is the probability of event  $A$  without knowledge about  
10344 whether  $B$  happens or not.
- 10345 • Pr( $B | A$ ) is the **likelihood**, the probability of event  $B$  given that event  $A$  has oc-  
10346 curred.
- 10347 • Pr( $A | B$ ) is the **posterior**, the probability of event  $A$  with knowledge that  $B$  has  
10348 occurred.

10349 **Example** The classic examples for Bayes' rule involve tests for rare diseases, but Man-  
10350 ning and Schütze (1999) reframe this example in a linguistic setting. Suppose that you are  
10351 interested in a rare syntactic construction, such as *parasitic gaps*, which occur on average  
10352 once in 100,000 sentences. Here is an example of a parasitic gap:

10353 (A.1) *Which class did you attend ... without registering for ...?*

10354 Lana Linguist has developed a complicated pattern matcher that attempts to identify  
10355 sentences with parasitic gaps. It's pretty good, but it's not perfect:

- 10356 • If a sentence has a parasitic gap, the pattern matcher will find it with probability  
10357 0.95. (This is the **recall**, which is one minus the **false positive rate**.)
- 10358 • If the sentence doesn't have a parasitic gap, the pattern matcher will wrongly say it  
10359 does with probability 0.005. (This is the **false positive rate**, which is one minus the  
10360 **precision**.)

10361 Suppose that Lana's pattern matcher says that a sentence contains a parasitic gap. What  
10362 is the probability that this is true?

Let  $G$  be the event of a sentence having a parasitic gap, and  $T$  be the event of the test being positive. We are interested in the probability of a sentence having a parasitic gap given that the test is positive. This is the conditional probability  $\Pr(G | T)$ , and it can be computed by Bayes' rule:

$$\Pr(G | T) = \frac{\Pr(T | G) \times \Pr(G)}{\Pr(T)}. \quad [\text{A.14}]$$

10363 We already know both terms in the numerator:  $\Pr(T | G)$  is the recall, which is 0.95;  $\Pr(G)$   
10364 is the prior, which is  $10^{-5}$ .

10365 We are not given the denominator, but it can be computed using tools developed earlier  
10366 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,

$$\Pr(G | T) = \frac{\Pr(T | G) \Pr(G)}{\Pr(T)} \quad [\text{A.20}]$$

$$= \frac{0.95 \times 10^{-5}}{0.95 \times 10^{-5} + 0.005 \times (1 - 10^{-5})} \quad [\text{A.21}]$$

$$\approx 0.002. \quad [\text{A.22}]$$

10367 Lana's pattern matcher seems accurate, with false positive and false negative rates  
10368 below 5%. Yet the extreme rarity of the phenomenon means that a positive result from the  
10369 detector is most likely to be wrong.

## 10370 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.

## 10374 A.4 Random variables

10375 Random variables are functions from events to  $\mathbb{R}^n$ , where  $\mathbb{R}$  is the set of real numbers.  
 10376 This subsumes several useful special cases:

- 10377 • An **indicator random variable** is a function from events to the set  $\{0, 1\}$ . In the coin  
 10378 flip example, we can define  $Y$  as an indicator random variable, taking the value  
 10379 1 when the coin has come up heads on at least one flip. This would include the  
 10380 outcomes  $\{HH, HT, TH\}$ . The probability  $\Pr(Y = 1)$  is the sum of the probabilities  
 10381 of these outcomes,  $\Pr(Y = 1) = \frac{1}{4} + \frac{1}{4} + \frac{1}{4} = \frac{3}{4}$ .
- 10382 • A **discrete random variable** is a function from events to a discrete subset of  $\mathbb{R}$ . Con-  
 10383 sider the coin flip example: the number of heads on two flips,  $X$ , can be viewed as a  
 10384 discrete random variable,  $X \in \{0, 1, 2\}$ . The event probability  $\Pr(X = 1)$  can again be  
 10385 computed as the sum of the probabilities of the events in which there is one head,  
 10386  $\{HT, TH\}$ , 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  $\Pr(X = x)$ .<sup>1</sup> The function  $p_X$  is called

<sup>1</sup>In general, capital letters (e.g.,  $X$ ) refer to random variables, and lower-case letters (e.g.,  $x$ ) refer to specific values. When the distribution is clear from context, I will simply write  $p(x)$ .

a probability **mass** function (pmf) if  $X$  is discrete; it is called a probability **density** function (pdf) if  $X$  is continuous. In either case, the function must sum to one, and all values must be non-negative:

$$\int_x p_X(x)dx = 1 \quad [A.28]$$

$$\forall x, p_X(x) \geq 0. \quad [A.29]$$

Probabilities over multiple random variables can written as **joint probabilities**, e.g.,  $p_{A,B}(a,b) = \Pr(A = a \cap B = b)$ . Several properties of event probabilities carry over to probability distributions over random variables:

- The **marginal probability distribution** is  $p_A(a) = \sum_b p_{A,B}(a,b)$ .
- The **conditional probability distribution** is  $p_{A|B}(a | b) = \frac{p_{A,B}(a,b)}{p_B(b)}$ .
- Random variables  $A$  and  $B$  are independent iff  $p_{A,B}(a,b) = p_A(a) \times p_B(b)$ .

## A.5 Expectations

Sometimes we want the **expectation** of a function, such as  $E[g(x)] = \sum_{x \in \mathcal{X}} g(x)p(x)$ . Expectations are easiest to think about in terms of probability distributions over discrete events:

- If it is sunny, Lucia will eat three ice creams.
- If it is rainy, she will eat only one ice cream.
- There's a 80% chance it will be sunny.
- The expected number of ice creams she will eat is  $0.8 \times 3 + 0.2 \times 1 = 2.6$ .

If the random variable  $X$  is continuous, the expectation is an integral:

$$E[g(x)] = \int_{\mathcal{X}} g(x)p(x)dx \quad [A.30]$$

For example, a fast food restaurant in Quebec has a special offer for cold days: they give a 1% discount on poutine for every degree below zero. Assuming a thermometer with infinite precision, the expected price would be an integral over all possible temperatures,

$$E[\text{price}(x)] = \int_{\mathcal{X}} \min(1, 1+x) \times \text{original-price} \times p(x)dx. \quad [A.31]$$

10405 **A.6 Modeling and estimation**

10406 **Probabilistic models** provide a principled way to reason about random events and ran-  
 10407 dom variables. Let's consider the coin toss example. Each toss can be modeled as a ran-  
 10408 dom event, with probability  $\theta$  of the event  $H$ , and probability  $1 - \theta$  of the complementary  
 10409 event  $T$ . If we write a random variable  $X$  as the total number of heads on three coin  
 10410 flips, then the distribution of  $X$  depends on  $\theta$ . In this case,  $X$  is distributed as a **binomial**  
 10411 **random variable**, meaning that it is drawn from a binomial distribution, with **parameters**  
 10412  $(\theta, N = 3)$ . This is written,

$$X \sim \text{Binomial}(\theta, N = 3). \quad [\text{A.32}]$$

10413 The properties of the binomial distribution enable us to make statements about the  $X$ ,  
 10414 such as its expected value and the likelihood that its value will fall within some interval.

Now suppose that  $\theta$  is unknown, but we have run an experiment, in which we ex-  
 ecuted  $N$  trials, and obtained  $x$  heads. We can **estimate**  $\theta$  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  $\theta$  and  $N$  are parameters of the probability function.  
 The likelihood  $p_X(x; \theta, N)$  can be computed from the binomial distribution,

$$p_X(x; \theta, N) = \frac{N!}{x!(N-x)!} \theta^x (1-\theta)^{N-x}. \quad [\text{A.34}]$$

10415 This likelihood is proportional to the product of the probability of individual out-  
 10416 comes: for example, the sequence  $T, H, H, T, H$  would have probability  $\theta^3(1-\theta)^2$ . The  
 10417 term  $\frac{N!}{x!(N-x)!}$  arises from the many possible orderings by which we could obtain  $x$  heads  
 10418 on  $N$  trials. This term does not depend on  $\theta$ , so it can be ignored during estimation.

In practice, we maximize the log-likelihood, which is a monotonic function of the like-  
 lihood. Under the binomial distribution, the log-likelihood is a **convex** function of  $\theta$  (see

§ 2.3), so it can be maximized by taking the derivative and setting it equal to zero.

$$\ell(\theta) = x \log \theta + (N - x) \log(1 - \theta) \quad [\text{A.35}]$$

$$\frac{\partial \ell(\theta)}{\partial \theta} = \frac{x}{\theta} - \frac{N - x}{1 - \theta} \quad [\text{A.36}]$$

$$\frac{N - x}{1 - \theta} = \frac{x}{\theta} \quad [\text{A.37}]$$

$$\frac{N - x}{x} = \frac{1 - \theta}{\theta} \quad [\text{A.38}]$$

$$\frac{N}{x} - 1 = \frac{1}{\theta} - 1 \quad [\text{A.39}]$$

$$\hat{\theta} = \frac{x}{N}. \quad [\text{A.40}]$$

10419 In this case, the maximum likelihood estimate is equal to  $\frac{x}{N}$ , the fraction of trials that  
 10420 came up heads. This intuitive solution is also known as the **relative frequency estimate**,  
 10421 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  $\theta$ . To incorporate this prior information, we can treat  $\theta$  as a random variable, and use Bayes' rule:

$$p(\theta | x; N) = \frac{p(x | \theta) \times p(\theta)}{p(x)} \quad [\text{A.41}]$$

$$\propto p(x | \theta) \times p(\theta) \quad [\text{A.42}]$$

$$\hat{\theta} = \operatorname{argmax}_{\theta} p(x | \theta) \times p(\theta). \quad [\text{A.43}]$$

10422 This is the **maximum a posteriori** (MAP) estimate. Given a form for  $p(\theta)$ , you can de-  
 10423 rive the MAP estimate using the same approach that was used to derive the maximum  
 10424 likelihood estimate.

## 10425 Additional resources

10426 A good introduction to probability theory is offered by Manning and Schütze (1999),  
 10427 which helped to motivate this section. For more detail, Sharon Goldwater provides an-  
 10428 other useful reference, <http://homepages.inf.ed.ac.uk/sgwater/teaching/general/probability.pdf>. A historical and philosophical perspective on probability is offered  
 10429 by Diaconis and Skyrms (2017).



10431 **Appendix B**

10432 **Numerical optimization**

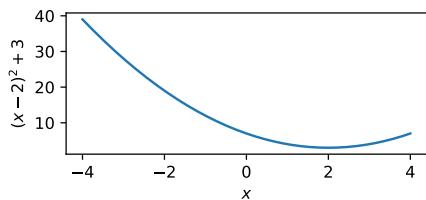
10433 Unconstrained numerical optimization involves solving problems of the form,

$$\min_{\mathbf{x} \in \mathbb{R}^D} f(\mathbf{x}), \quad [\text{B.1}]$$

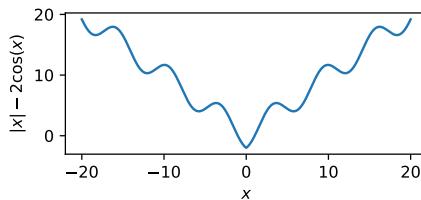
10434 where  $\mathbf{x} \in \mathbb{R}^D$  is a vector of  $D$  real numbers.

10435 Differentiation is fundamental to continuous optimization. Suppose that at some  $\mathbf{x}^*$ ,  
10436 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**  
10437 **point** of  $f$ . For a **convex** function  $f$  (defined in § 2.3),  $f(\mathbf{x}^*)$  is equal to the global minimum  
10438 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,

$$\frac{\partial d}{\partial x_1} = x_1 - 2 \quad [B.2]$$

$$\frac{\partial d}{\partial x_2} = x_2 - 1 \quad [B.3]$$

10439 The unique minimum is  $\mathbf{x}^* = [2, 1]^\top$ .

10440 For non-convex functions, critical points are not necessarily global minima. A **local**  
 10441 **minimum**  $\mathbf{x}^*$  is a point at which the function takes a smaller value than at all nearby  
 10442 neighbors: formally,  $\mathbf{x}^*$  is a local minimum if there is some positive  $\epsilon$  such that  $f(\mathbf{x}^*) \leq$   
 10443  $f(\mathbf{x})$  for all  $\mathbf{x}$  within distance  $\epsilon$  of  $\mathbf{x}^*$ . Figure B.1b shows the function  $f(x) = |x| - 2 \cos(x)$ ,  
 10444 which has many local minima, as well as a unique global minimum at  $x = 0$ . A critical  
 10445 point may also be the local or global maximum of the function; it may be a **saddle point**,  
 10446 which is a minimum with respect to at least one coordinate, and a maximum with respect  
 10447 to at least one other coordinate; it may be an **inflection point**, which is neither a minimum  
 10448 nor maximum. When available, the second derivative of  $f$  can help to distinguish these  
 10449 cases.

## 10450 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,

$$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} - \eta^{(t)} \nabla_{\mathbf{x}^{(t)}} f. \quad [B.4]$$

10451 where  $\eta^{(t)} > 0$  is a **step size**. If the step size is chosen appropriately, this procedure will  
 10452 find the global minimum of a differentiable convex function. For non-convex functions,  
 10453 gradient descent will find a local minimum. The extension to non-differentiable convex  
 10454 functions is discussed in § 2.3.

## 10455 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  $\lambda_c$  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.<sup>1</sup> 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,

$$\ell^{(i)}(\boldsymbol{\theta}) = 1 - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) + \max_{y \neq y^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y). \quad [\text{B.13}]$$

---

<sup>1</sup>This is the basis for the name of the algorithm: it is passive when the loss is zero, but it aggressively moves to make the loss zero when necessary.

When the margin loss is zero for  $\theta^{(i-1)}$ , the optimal solution is simply to set  $\theta^* = \theta^{(i-1)}$ , so we will focus on the case where  $\ell^{(i)}(\theta^{(i-1)}) > 0$ . The Lagrangian for this problem is,

$$L(\theta, \lambda) = \frac{1}{2} \|\theta - \theta^{(i-1)}\|^2 + \lambda \ell^{(i)}(\theta), \quad [\text{B.14}]$$

Holding  $\lambda$  constant, we can solve for  $\theta$  by differentiating,

$$\nabla_{\theta} L = \theta - \theta^{(i-1)} + \lambda \frac{\partial}{\partial \theta} \ell^{(i)}(\theta) \quad [\text{B.15}]$$

$$\theta^* = \theta^{(i-1)} + \lambda \delta, \quad [\text{B.16}]$$

where  $\delta = f(x^{(i)}, y^{(i)}) - f(x^{(i)}, \hat{y})$  and  $\hat{y} = \operatorname{argmax}_{y \neq y^{(i)}} \theta \cdot f(x^{(i)}, y)$ .

The Lagrange multiplier  $\lambda$  acts as the learning rate in a perceptron-style update to  $\theta$ . We can solve for  $\lambda$  by plugging  $\theta^*$  back into the Lagrangian, obtaining the dual function,

$$D(\lambda) = \frac{1}{2} \|\theta^{(i-1)} + \lambda \delta - \theta^{(i-1)}\|^2 + \lambda(1 - (\theta^{(i-1)} + \lambda \delta) \cdot \delta) \quad [\text{B.17}]$$

$$= \frac{\lambda^2}{2} \|\delta\|^2 - \lambda^2 \|\delta\|^2 + \lambda(1 - \theta^{(i-1)} \cdot \delta) \quad [\text{B.18}]$$

$$= -\frac{\lambda^2}{2} \|\delta\|^2 + \lambda \ell^{(i)}(\theta^{(i-1)}). \quad [\text{B.19}]$$

Differentiating and solving for  $\lambda$ ,

$$\frac{\partial D}{\partial \lambda} = -\lambda \|\delta\|^2 + \ell^{(i)}(\theta^{(i-1)}) \quad [\text{B.20}]$$

$$\lambda^* = \frac{\ell^{(i)}(\theta^{(i-1)})}{\|\delta\|^2}. \quad [\text{B.21}]$$

The complete update equation is therefore:

$$\theta^* = \theta^{(i-1)} + \frac{\ell^{(i)}(\theta^{(i-1)})}{\|f(x^{(i)}, y^{(i)}) - f(x^{(i)}, \hat{y})\|^2} (f(x^{(i)}, y^{(i)}) - f(x^{(i)}, \hat{y})). \quad [\text{B.22}]$$

This update has strong intuitive support. The numerator of the learning rate grows with the loss. The denominator grows with the norm of the difference between the feature vectors associated with the correct and predicted label. If this norm is large, then the step with respect to each feature should be small, and vice versa.

10473

# Bibliography

- 10474 Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis,  
10475 J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia,  
10476 R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore,  
10477 D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A.  
10478 Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Watten-  
10479 berg, M. Wicke, Y. Yu, and X. Zheng (2016). Tensorflow: Large-scale machine learning  
10480 on heterogeneous distributed systems. *CoRR abs/1603.04467*.
- 10481 Abend, O. and A. Rappoport (2017). The state of the art in semantic representation. In  
10482 *Proceedings of the Association for Computational Linguistics (ACL)*.
- 10483 Abney, S., R. E. Schapire, and Y. Singer (1999). Boosting applied to tagging and PP attach-  
10484 ment. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.  
10485 132–134.
- 10486 Abney, S. P. (1987). *The English noun phrase in its sentential aspect*. Ph. D. thesis, Mas-  
10487 sachusetts Institute of Technology.
- 10488 Abney, S. P. and M. Johnson (1991). Memory requirements and local ambiguities of pars-  
10489 ing strategies. *Journal of Psycholinguistic Research* 20(3), 233–250.
- 10490 Adafre, S. F. and M. De Rijke (2006). Finding similar sentences across multiple languages  
10491 in wikipedia. In *Proceedings of the Workshop on NEW TEXT Wikis and blogs and other*  
10492 *dynamic text sources*.
- 10493 Ahn, D. (2006). The stages of event extraction. In *Proceedings of the Workshop on Annotating*  
10494 *and Reasoning about Time and Events*, pp. 1–8. Association for Computational Linguistics.
- 10495 Aho, A. V., M. S. Lam, R. Sethi, and J. D. Ullman (2006). Compilers: Principles, techniques,  
10496 & tools.
- 10497 Aikhenvald, A. Y. (2004). *Evidentiality*. Oxford University Press.

- 10498 Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on  
10499 Automatic Control* 19(6), 716–723.
- 10500 Akmajian, A., R. A. Demers, A. K. Farmer, and R. M. Harnish (2010). *Linguistics: An  
10501 introduction to language and communication* (Sixth ed.). Cambridge, MA: MIT press.
- 10502 Alfau, F. (1999). *Chromos*. Dalkey Archive Press.
- 10503 Allauzen, C., M. Riley, J. Schalkwyk, W. Skut, and M. Mohri (2007). OpenFst: A gen-  
10504 eral and efficient weighted finite-state transducer library. In *International Conference on  
10505 Implementation and Application of Automata*, pp. 11–23. Springer.
- 10506 Allen, J. F. (1984). Towards a general theory of action and time. *Artificial intelligence* 23(2),  
10507 123–154.
- 10508 Allen, J. F., B. W. Miller, E. K. Ringger, and T. Sikorski (1996). A robust system for natural  
10509 spoken dialogue. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.  
10510 62–70.
- 10511 Allen, J. F., L. K. Schubert, G. Ferguson, P. Heeman, C. H. Hwang, T. Kato, M. Light,  
10512 N. Martin, B. Miller, M. Poesio, and D. Traum (1995). The TRAINS project: A case  
10513 study in building a conversational planning agent. *Journal of Experimental & Theoretical  
10514 Artificial Intelligence* 7(1), 7–48.
- 10515 Alm, C. O., D. Roth, and R. Sproat (2005). Emotions from text: machine learning for  
10516 text-based emotion prediction. In *Proceedings of Empirical Methods for Natural Language  
10517 Processing (EMNLP)*, pp. 579–586.
- 10518 Aluísio, S., J. Pelizzoni, A. Marchi, L. de Oliveira, R. Manenti, and V. Marquiafável (2003).  
10519 An account of the challenge of tagging a reference corpus for Brazilian Portuguese.  
10520 *Computational Processing of the Portuguese Language*, 194–194.
- 10521 Anand, P., M. Walker, R. Abbott, J. E. Fox Tree, R. Bowman, and M. Minor (2011). Cats rule  
10522 and dogs drool!: Classifying stance in online debate. In *Proceedings of the 2nd Workshop  
10523 on Computational Approaches to Subjectivity and Sentiment Analysis*, Portland, Oregon, pp.  
10524 1–9. Association for Computational Linguistics.
- 10525 Anandkumar, A. and R. Ge (2016). Efficient approaches for escaping higher order saddle  
10526 points in non-convex optimization. In *Proceedings of the Conference On Learning Theory  
10527 (COLT)*, pp. 81–102.
- 10528 Anandkumar, A., R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky (2014). Tensor decompo-  
10529 sitions for learning latent variable models. *The Journal of Machine Learning Research* 15(1),  
10530 2773–2832.

- 10531 Ando, R. K. and T. Zhang (2005). A framework for learning predictive structures from  
10532 multiple tasks and unlabeled data. *The Journal of Machine Learning Research* 6, 1817–  
10533 1853.
- 10534 Andor, D., C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and  
10535 M. Collins (2016). Globally normalized transition-based neural networks. In *Proceedings*  
10536 of the Association for Computational Linguistics (ACL), pp. 2442–2452.
- 10537 Angeli, G., P. Liang, and D. Klein (2010). A simple domain-independent probabilistic ap-  
10538 proach to generation. In *Proceedings of Empirical Methods for Natural Language Processing*  
10539 (EMNLP), pp. 502–512.
- 10540 Antol, S., A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh  
10541 (2015). Vqa: Visual question answering. In *Proceedings of the International Conference on*  
10542 *Computer Vision (ICCV)*, pp. 2425–2433.
- 10543 Aronoff, M. (1976). *Word formation in generative grammar*. MIT Press.
- 10544 Arora, S. and B. Barak (2009). *Computational complexity: a modern approach*. Cambridge  
10545 University Press.
- 10546 Arora, S., R. Ge, Y. Halpern, D. Mimmo, A. Moitra, D. Sontag, Y. Wu, and M. Zhu (2013).  
10547 A practical algorithm for topic modeling with provable guarantees. In *Proceedings of the*  
10548 *International Conference on Machine Learning (ICML)*, pp. 280–288.
- 10549 Arora, S., Y. Li, Y. Liang, T. Ma, and A. Risteski (2016). Linear algebraic structure of word  
10550 senses, with applications to polysemy. *arXiv preprint arXiv:1601.03764*.
- 10551 Artstein, R. and M. Poesio (2008). Inter-coder agreement for computational linguistics.  
10552 *Computational Linguistics* 34(4), 555–596.
- 10553 Artzi, Y. and L. Zettlemoyer (2013). Weakly supervised learning of semantic parsers for  
10554 mapping instructions to actions. *Transactions of the Association for Computational Linguis-*  
10555 *tics* 1, 49–62.
- 10556 Attardi, G. (2006). Experiments with a multilanguage non-projective dependency parser.  
10557 In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 166–170.
- 10558 Auer, P. (2013). *Code-switching in conversation: Language, interaction and identity*. Routledge.
- 10559 Auer, S., C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives (2007). Dbpedia: A  
10560 nucleus for a web of open data. *The semantic web*, 722–735.
- 10561 Austin, J. L. (1962). *How to do things with words*. Oxford University Press.

- 10562 Aw, A., M. Zhang, J. Xiao, and J. Su (2006). A phrase-based statistical model for SMS text  
10563 normalization. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.  
10564 33–40.
- 10565 Ba, J. L., J. R. Kiros, and G. E. Hinton (2016). Layer normalization. *arXiv preprint*  
10566 *arXiv:1607.06450*.
- 10567 Bagga, A. and B. Baldwin (1998a). Algorithms for scoring coreference chains. In *Proceed-  
10568 ings of the Language Resources and Evaluation Conference*, pp. 563–566.
- 10569 Bagga, A. and B. Baldwin (1998b). Entity-based cross-document coreferencing using the  
10570 vector space model. In *Proceedings of the International Conference on Computational Lin-  
10571 guistics (COLING)*, pp. 79–85.
- 10572 Bahdanau, D., K. Cho, and Y. Bengio (2014). Neural machine translation by jointly learn-  
10573 ing to align and translate. In *Neural Information Processing Systems (NIPS)*.
- 10574 Baldwin, T. and S. N. Kim (2010). Multiword expressions. In *Handbook of natural language  
10575 processing*, Volume 2, pp. 267–292. Boca Raton, USA: CRC Press.
- 10576 Balle, B., A. Quattoni, and X. Carreras (2011). A spectral learning algorithm for finite state  
10577 transducers. In *Proceedings of the European Conference on Machine Learning and Principles  
10578 and Practice of Knowledge Discovery in Databases (ECML)*, pp. 156–171.
- 10579 Banarescu, L., C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight,  
10580 P. Koehn, M. Palmer, and N. Schneider (2013, August). Abstract meaning represen-  
10581 tation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability  
10582 with Discourse*, Sofia, Bulgaria, pp. 178–186. Association for Computational  
10583 Linguistics.
- 10584 Banko, M., M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni (2007). Open  
10585 information extraction from the web. In *Proceedings of the International Joint Conference  
10586 on Artificial Intelligence (IJCAI)*, pp. 2670–2676.
- 10587 Bansal, N., A. Blum, and S. Chawla (2004). Correlation clustering. *Machine Learning* 56(1–  
10588 3), 89–113.
- 10589 Barber, D. (2012). *Bayesian reasoning and machine learning*. Cambridge University Press.
- 10590 Barman, U., A. Das, J. Wagner, and J. Foster (2014, October). Code mixing: A challenge for  
10591 language identification in the language of social media. In *Proceedings of the First Work-  
10592 shop on Computational Approaches to Code Switching*, Doha, Qatar, pp. 13–23. Association  
10593 for Computational Linguistics.

- 10594 Barnickel, T., J. Weston, R. Collobert, H.-W. Mewes, and V. Stümpflen (2009). Large scale  
10595 application of neural network based semantic role labeling for automated relation ex-  
10596 traction from biomedical texts. *PLoS One* 4(7), e6393.
- 10597 Baron, A. and P. Rayson (2008). Vard2: A tool for dealing with spelling variation in his-  
10598 torical corpora. In *Postgraduate conference in corpus linguistics*.
- 10599 Baroni, M., R. Bernardi, and R. Zamparelli (2014). Frege in space: A program for compo-  
10600 sitional distributional semantics. *Linguistic Issues in Language Technologies*.
- 10601 Barzilay, R. and M. Lapata (2008, mar). Modeling local coherence: An Entity-Based ap-  
10602 proach. *Computational Linguistics* 34(1), 1–34.
- 10603 Barzilay, R. and K. R. McKeown (2005). Sentence fusion for multidocument news summa-  
10604 rization. *Computational Linguistics* 31(3), 297–328.
- 10605 Beesley, K. R. and L. Karttunen (2003). *Finite-state morphology*. Stanford, CA: Center for  
10606 the Study of Language and Information.
- 10607 Bejan, C. A. and S. Harabagiu (2014). Unsupervised event coreference resolution. *Compu-*  
10608 *tational Linguistics* 40(2), 311–347.
- 10609 Bell, E. T. (1934). Exponential numbers. *The American Mathematical Monthly* 41(7), 411–419.
- 10610 Bender, E. M. (2013, jun). *Linguistic Fundamentals for Natural Language Processing: 100*  
10611 *Essentials from Morphology and Syntax*, Volume 6 of *Synthesis Lectures on Human Language*  
10612 *Technologies*. Morgan & Claypool Publishers.
- 10613 Bengio, S., O. Vinyals, N. Jaitly, and N. Shazeer (2015). Scheduled sampling for sequence  
10614 prediction with recurrent neural networks. In *Neural Information Processing Systems*  
10615 (*NIPS*), pp. 1171–1179.
- 10616 Bengio, Y., R. Ducharme, P. Vincent, and C. Janvin (2003). A neural probabilistic language  
10617 model. *The Journal of Machine Learning Research* 3, 1137–1155.
- 10618 Bengio, Y., P. Simard, and P. Frasconi (1994). Learning long-term dependencies with gra-  
10619 dient descent is difficult. *IEEE Transactions on Neural Networks* 5(2), 157–166.
- 10620 Bengtsson, E. and D. Roth (2008). Understanding the value of features for coreference  
10621 resolution. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,  
10622 pp. 294–303.
- 10623 Benjamini, Y. and Y. Hochberg (1995). Controlling the false discovery rate: a practical and  
10624 powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B*  
10625 (*Methodological*), 289–300.

- 10626 Berant, J., A. Chou, R. Frostig, and P. Liang (2013). Semantic parsing on freebase from  
10627 question-answer pairs. In *Proceedings of Empirical Methods for Natural Language Processing* (EMNLP), pp. 1533–1544.
- 10629 Berant, J., V. Srikumar, P.-C. Chen, A. Vander Linden, B. Harding, B. Huang, P. Clark, and  
10630 C. D. Manning (2014). Modeling biological processes for reading comprehension. In  
10631 *Proceedings of Empirical Methods for Natural Language Processing* (EMNLP).
- 10632 Berg-Kirkpatrick, T., A. Bouchard-Côté, J. DeNero, and D. Klein (2010). Painless unsuper-  
10633 vised learning with features. In *Proceedings of the North American Chapter of the Associa-*  
10634 *tion for Computational Linguistics* (NAACL), pp. 582–590.
- 10635 Berg-Kirkpatrick, T., D. Burkett, and D. Klein (2012). An empirical investigation of sta-  
10636 tistical significance in NLP. In *Proceedings of Empirical Methods for Natural Language*  
10637 *Processing* (EMNLP), pp. 995–1005.
- 10638 Berger, A. L., V. J. D. Pietra, and S. A. D. Pietra (1996). A maximum entropy approach to  
10639 natural language processing. *Computational linguistics* 22(1), 39–71.
- 10640 Bergsma, S., D. Lin, and R. Goebel (2008). Distributional identification of non-referential  
10641 pronouns. In *Proceedings of the Association for Computational Linguistics* (ACL), pp. 10–18.
- 10642 Bernardi, R., R. Cakici, D. Elliott, A. Erdem, E. Erdem, N. Ikizler-Cinbis, F. Keller, A. Mus-  
10643 cat, and B. Plank (2016). Automatic description generation from images: A survey of  
10644 models, datasets, and evaluation measures. *Journal of Artificial Intelligence Research* 55,  
10645 409–442.
- 10646 Bertsekas, D. P. (2012). Incremental gradient, subgradient, and proximal methods for  
10647 convex optimization: A survey. See Sra et al. (2012).
- 10648 Bhatia, P., R. Guthrie, and J. Eisenstein (2016). Morphological priors for probabilistic neu-  
10649 ral word embeddings. In *Proceedings of Empirical Methods for Natural Language Processing*  
10650 (EMNLP).
- 10651 Bhatia, P., Y. Ji, and J. Eisenstein (2015). Better document-level sentiment analysis from  
10652 first discourse parsing. In *Proceedings of Empirical Methods for Natural Language Processing*  
10653 (EMNLP).
- 10654 Biber, D. (1991). *Variation across speech and writing*. Cambridge University Press.
- 10655 Bird, S., E. Klein, and E. Loper (2009). *Natural language processing with Python*. California:  
10656 O'Reilly Media.
- 10657 Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

- 10658 Björkelund, A. and P. Nugues (2011). Exploring lexicalized features for coreference reso-  
10659 lution. In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 45–50.
- 10660 Blackburn, P. and J. Bos (2005). *Representation and inference for natural language: A first*  
10661 *course in computational semantics*. CSLI.
- 10662 Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM* 55(4), 77–84.
- 10663 Blei, D. M. (2014). Build, compute, critique, repeat: Data analysis with latent variable  
10664 models. *Annual Review of Statistics and Its Application* 1, 203–232.
- 10665 Blei, D. M., A. Y. Ng, and M. I. Jordan (2003). Latent dirichlet allocation. *the Journal of*  
10666 *machine Learning research* 3, 993–1022.
- 10667 Blitzer, J., M. Dredze, and F. Pereira (2007). Biographies, bollywood, boom-boxes and  
10668 blenders: Domain adaptation for sentiment classification. In *Proceedings of the Associa-*  
10669 *tion for Computational Linguistics (ACL)*, pp. 440–447.
- 10670 Blum, A. and T. Mitchell (1998). Combining labeled and unlabeled data with co-training.  
10671 In *Proceedings of the Conference On Learning Theory (COLT)*, pp. 92–100.
- 10672 Bobrow, D. G., R. M. Kaplan, M. Kay, D. A. Norman, H. Thompson, and T. Winograd  
10673 (1977). Gus, a frame-driven dialog system. *Artificial intelligence* 8(2), 155–173.
- 10674 Bochnet, B. (2010). Very high accuracy and fast dependency parsing is not a contradiction.  
10675 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.  
10676 89–97.
- 10677 Boitet, C. (1988). Pros and cons of the pivot and transfer approaches in multilingual ma-  
10678 chine translation. *Readings in machine translation*, 273–279.
- 10679 Bojanowski, P., E. Grave, A. Joulin, and T. Mikolov (2017). Enriching word vectors with  
10680 subword information. *Transactions of the Association for Computational Linguistics* 5, 135–  
10681 146.
- 10682 Bollacker, K., C. Evans, P. Paritosh, T. Sturge, and J. Taylor (2008). Freebase: a collabora-  
10683 tively created graph database for structuring human knowledge. In *Proceedings of the*  
10684 *ACM International Conference on Management of Data (SIGMOD)*, pp. 1247–1250. AcM.
- 10685 Bolukbasi, T., K.-W. Chang, J. Y. Zou, V. Saligrama, and A. T. Kalai (2016). Man is to  
10686 computer programmer as woman is to homemaker? debiasing word embeddings. In *Neural Information Processing Systems (NIPS)*, pp. 4349–4357.
- 10688 Bordes, A., N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko (2013). Translating  
10689 embeddings for modeling multi-relational data. In *Neural Information Processing Systems*  
10690 (*NIPS*), pp. 2787–2795.

- 10691 Bordes, A., J. Weston, R. Collobert, Y. Bengio, et al. (2011). Learning structured embed-  
 10692       dings of knowledge bases. In *Proceedings of the National Conference on Artificial Intelligence  
 10693 (AAAI)*, pp. 301–306.
- 10694 Borges, J. L. (1993). *Other Inquisitions 1937–1952*. University of Texas Press. Translated by  
 10695       Ruth L. C. Simms.
- 10696 Botha, J. A. and P. Blunsom (2014). Compositional morphology for word representations  
 10697       and language modelling. In *Proceedings of the International Conference on Machine Learn-  
 10698 ing (ICML)*.
- 10699 Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*,  
 10700       pp. 421–436. Springer.
- 10701 Bottou, L., F. E. Curtis, and J. Nocedal (2016). Optimization methods for large-scale ma-  
 10702       chine learning. *arXiv preprint arXiv:1606.04838*.
- 10703 Bowman, S. R., L. Vilnis, O. Vinyals, A. Dai, R. Jozefowicz, and S. Bengio (2016). Gen-  
 10704       erating sentences from a continuous space. In *Proceedings of the Conference on Natural  
 10705 Language Learning (CoNLL)*, pp. 10–21.
- 10706 boyd, d. and K. Crawford (2012). Critical questions for big data. *Information, Communica-  
 10707 tion & Society* 15(5), 662–679.
- 10708 Boyd, S. and L. Vandenberghe (2004). *Convex Optimization*. New York: Cambridge Uni-  
 10709       versity Press.
- 10710 Branavan, S., H. Chen, J. Eisenstein, and R. Barzilay (2009). Learning document-level  
 10711       semantic properties from free-text annotations. *Journal of Artificial Intelligence Re-  
 10712 search* 34(2), 569–603.
- 10713 Branavan, S. R., H. Chen, L. S. Zettlemoyer, and R. Barzilay (2009). Reinforcement learning  
 10714       for mapping instructions to actions. In *Proceedings of the Association for Computational  
 10715 Linguistics (ACL)*, pp. 82–90.
- 10716 Braud, C., O. Lacroix, and A. Søgaard (2017). Does syntax help discourse segmenta-  
 10717       tion? not so much. In *Proceedings of Empirical Methods for Natural Language Processing  
 10718 (EMNLP)*, pp. 2432–2442.
- 10719 Briscoe, T. (2011). Introduction to formal semantics for natural language.
- 10720 Brown, P. F., J. Cocke, S. A. D. Pietra, V. J. D. Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer,  
 10721       and P. S. Roossin (1990). A statistical approach to machine translation. *Computational  
 10722 linguistics* 16(2), 79–85.

- 10723 Brown, P. F., P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai (1992). Class-based  
10724 n-gram models of natural language. *Computational linguistics* 18(4), 467–479.
- 10725 Brown, P. F., V. J. D. Pietra, S. A. D. Pietra, and R. L. Mercer (1993). The mathematics  
10726 of statistical machine translation: Parameter estimation. *Computational linguistics* 19(2),  
10727 263–311.
- 10728 Brun, C. and C. Roux (2014). Décomposition des “hash tags” pour l’amélioration de la  
10729 classification en polarité des “tweets”. *Proceedings of Traitement Automatique des Langues  
10730 Naturelles*, 473–478.
- 10731 Bruni, E., N.-K. Tran, and M. Baroni (2014). Multimodal distributional semantics. *Journal  
10732 of Artificial Intelligence Research* 49(2014), 1–47.
- 10733 Bullinaria, J. A. and J. P. Levy (2007). Extracting semantic representations from word co-  
10734 occurrence statistics: A computational study. *Behavior research methods* 39(3), 510–526.
- 10735 Bunescu, R. C. and R. J. Mooney (2005). A shortest path dependency kernel for relation  
10736 extraction. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,  
10737 pp. 724–731.
- 10738 Bunescu, R. C. and M. Pasca (2006). Using encyclopedic knowledge for named entity  
10739 disambiguation. In *Proceedings of the European Chapter of the Association for Computational  
10740 Linguistics (EACL)*, pp. 9–16.
- 10741 Burstein, J., D. Marcu, and K. Knight (2003). Finding the WRITE stuff: Automatic identi-  
10742 fication of discourse structure in student essays. *IEEE Intelligent Systems* 18(1), 32–39.
- 10743 Burstein, J., J. Tetreault, and S. Andreyev (2010). Using entity-based features to model  
10744 coherence in student essays. In *Human language technologies: The 2010 annual conference  
10745 of the North American chapter of the Association for Computational Linguistics*, pp. 681–684.  
10746 Association for Computational Linguistics.
- 10747 Burstein, J., J. Tetreault, and M. Chodorow (2013). Holistic discourse coherence annotation  
10748 for noisy essay writing. *Dialogue & Discourse* 4(2), 34–52.
- 10749 Cai, Q. and A. Yates (2013). Large-scale semantic parsing via schema matching and lexicon  
10750 extension. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 423–  
10751 433.
- 10752 Caliskan, A., J. J. Bryson, and A. Narayanan (2017). Semantics derived automatically from  
10753 language corpora contain human-like biases. *Science* 356(6334), 183–186.
- 10754 Canny, J. (1987). A computational approach to edge detection. In *Readings in Computer  
10755 Vision*, pp. 184–203. Elsevier.

- 10756 Cappé, O. and E. Moulines (2009). On-line expectation–maximization algorithm for latent  
10757 data models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 71(3),  
10758 593–613.
- 10759 Carbonell, J. and J. Goldstein (1998). The use of mmr, diversity-based reranking for re-  
10760 ordering documents and producing summaries. In *Proceedings of ACM SIGIR conference*  
10761 on Research and development in information retrieval, pp. 335–336.
- 10762 Carbonell, J. R. (1970). Mixed-initiative man-computer instructional dialogues. Technical  
10763 report, BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS.
- 10764 Cardie, C. and K. Wagstaff (1999). Noun phrase coreference as clustering. In *Proceedings*  
10765 of *Empirical Methods for Natural Language Processing (EMNLP)*, pp. 82–89.
- 10766 Carletta, J. (1996). Assessing agreement on classification tasks: the kappa statistic. *Com-  
10767 putational linguistics* 22(2), 249–254.
- 10768 Carletta, J. (2007). Unleashing the killer corpus: experiences in creating the multi-  
10769 everything ami meeting corpus. *Language Resources and Evaluation* 41(2), 181–190.
- 10770 Carlson, L. and D. Marcu (2001). Discourse tagging reference manual. Technical Report  
10771 ISI-TR-545, Information Sciences Institute.
- 10772 Carlson, L., M. E. Okurowski, and D. Marcu (2002). RST discourse treebank. Linguistic  
10773 Data Consortium, University of Pennsylvania.
- 10774 Carpenter, B. (1997). *Type-logical semantics*. Cambridge, MA: MIT Press.
- 10775 Carreras, X., M. Collins, and T. Koo (2008). Tag, dynamic programming, and the percep-  
10776 tron for efficient, feature-rich parsing. In *Proceedings of the Conference on Natural Language*  
10777 *Learning (CoNLL)*, pp. 9–16.
- 10778 Carreras, X. and L. Màrquez (2005). Introduction to the conll-2005 shared task: Semantic  
10779 role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language*  
10780 *Learning*, pp. 152–164. Association for Computational Linguistics.
- 10781 Carroll, L. (1917). *Through the looking glass: And what Alice found there*. Chicago: Rand,  
10782 McNally.
- 10783 Chambers, N. and D. Jurafsky (2008). Jointly combining implicit constraints improves  
10784 temporal ordering. In *Proceedings of Empirical Methods for Natural Language Processing*  
10785 (*EMNLP*), pp. 698–706.
- 10786 Chang, K.-W., A. Krishnamurthy, A. Agarwal, H. Daume III, and J. Langford (2015).  
10787 Learning to search better than your teacher. In *Proceedings of the International Confer-  
10788 ence on Machine Learning (ICML)*.

- 10789 Chang, M.-W., L. Ratinov, and D. Roth (2007). Guiding semi-supervision with constraint-  
10790 driven learning. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.  
10791 280–287.
- 10792 Chang, M.-W., L.-A. Ratinov, N. Rizzolo, and D. Roth (2008). Learning and inference with  
10793 constraints. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp.  
10794 1513–1518.
- 10795 Chapman, W. W., W. Bridewell, P. Hanbury, G. F. Cooper, and B. G. Buchanan (2001). A  
10796 simple algorithm for identifying negated findings and diseases in discharge summaries.  
10797 *Journal of biomedical informatics* 34(5), 301–310.
- 10798 Charniak, E. (1997). Statistical techniques for natural language parsing. *AI magazine* 18(4),  
10799 33–43.
- 10800 Charniak, E. and M. Johnson (2005). Coarse-to-fine n-best parsing and maxent discrimi-  
10801 native reranking. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.  
10802 173–180.
- 10803 Chelba, C. and A. Acero (2006). Adaptation of maximum entropy capitalizer: Little data  
10804 can help a lot. *Computer Speech & Language* 20(4), 382–399.
- 10805 Chelba, C., T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson (2013).  
10806 One billion word benchmark for measuring progress in statistical language modeling.  
10807 *arXiv preprint arXiv:1312.3005*.
- 10808 Chen, D., J. Bolton, and C. D. Manning (2016). A thorough examination of the CNN/Daily  
10809 Mail reading comprehension task. In *Proceedings of the Association for Computational  
10810 Linguistics (ACL)*.
- 10811 Chen, D. and C. D. Manning (2014). A fast and accurate dependency parser using neural  
10812 networks. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,  
10813 pp. 740–750.
- 10814 Chen, D. L. and R. J. Mooney (2008). Learning to sportscast: a test of grounded language  
10815 acquisition. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp.  
10816 128–135.
- 10817 Chen, H., S. Branavan, R. Barzilay, and D. R. Karger (2009). Content modeling using latent  
10818 permutations. *Journal of Artificial Intelligence Research* 36(1), 129–163.
- 10819 Chen, M., Z. Xu, K. Weinberger, and F. Sha (2012). Marginalized denoising autoencoders  
10820 for domain adaptation. In *Proceedings of the International Conference on Machine Learning  
10821 (ICML)*.

- 10822 Chen, M. X., O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones, N. Parmar,  
 10823 M. Schuster, Z. Chen, Y. Wu, and M. Hughes (2018). The best of both worlds: Combin-  
 10824 ing recent advances in neural machine translation. In *Proceedings of the Association for*  
 10825 *Computational Linguistics (ACL)*.
- 10826 Chen, S. F. and J. Goodman (1999). An empirical study of smoothing techniques for lan-  
 10827 guage modeling. *Computer Speech & Language* 13(4), 359–393.
- 10828 Chen, T. and C. Guestrin (2016). Xgboost: A scalable tree boosting system. In *Proceedings*  
 10829 *of Knowledge Discovery and Data Mining (KDD)*, pp. 785–794.
- 10830 Chen, X., X. Qiu, C. Zhu, P. Liu, and X. Huang (2015). Long short-term memory neural  
 10831 networks for chinese word segmentation. In *Proceedings of Empirical Methods for Natural*  
 10832 *Language Processing (EMNLP)*, pp. 1197–1206.
- 10833 Chen, Y., S. Gilroy, A. Malletti, K. Knight, and J. May (2018). Recurrent neural networks  
 10834 as weighted language recognizers. In *Proceedings of the North American Chapter of the*  
 10835 *Association for Computational Linguistics (NAACL)*.
- 10836 Chen, Z. and H. Ji (2009). Graph-based event coreference resolution. In *Proceedings of*  
 10837 *the 2009 Workshop on Graph-based Methods for Natural Language Processing*, pp. 54–57.  
 10838 Association for Computational Linguistics.
- 10839 Cheng, X. and D. Roth (2013). Relational inference for wikification. In *Proceedings of*  
 10840 *Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1787–1796.
- 10841 Chiang, D. (2007). Hierarchical phrase-based translation. *Computational Linguistics* 33(2),  
 10842 201–228.
- 10843 Chiang, D., J. Graehl, K. Knight, A. Pauls, and S. Ravi (2010). Bayesian inference for  
 10844 finite-state transducers. In *Proceedings of the North American Chapter of the Association for*  
 10845 *Computational Linguistics (NAACL)*, pp. 447–455.
- 10846 Cho, K. (2015). Natural language understanding with distributed representation.  
 10847 *CoRR abs/1511.07916*.
- 10848 Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and  
 10849 Y. Bengio (2014). Learning phrase representations using rnn encoder-decoder for sta-  
 10850 tistical machine translation. In *Proceedings of Empirical Methods for Natural Language*  
 10851 *Processing (EMNLP)*.
- 10852 Chomsky, N. (1957). *Syntactic structures*. The Hague: Mouton & Co.
- 10853 Chomsky, N. (1982). *Some concepts and consequences of the theory of government and binding*,  
 10854 Volume 6. MIT press.

- 10855 Choromanska, A., M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun (2015). The loss  
10856 surfaces of multilayer networks. In *Proceedings of Artificial Intelligence and Statistics (AIS-*  
10857 *TATS)*, pp. 192–204.
- 10858 Christensen, J., S. Soderland, O. Etzioni, et al. (2010). Semantic role labeling for open  
10859 information extraction. In *Proceedings of the Workshop on Formalisms and Methodology for*  
10860 *Learning by Reading*, pp. 52–60. Association for Computational Linguistics.
- 10861 Christodoulopoulos, C., S. Goldwater, and M. Steedman (2010). Two decades of unsuper-  
10862 vised pos induction: How far have we come? In *Proceedings of Empirical Methods for*  
10863 *Natural Language Processing (EMNLP)*, pp. 575–584.
- 10864 Chu, Y.-J. and T.-H. Liu (1965). On shortest arborescence of a directed graph. *Scientia*  
10865 *Sinica* 14(10), 1396–1400.
- 10866 Chung, C. and J. W. Pennebaker (2007). The psychological functions of function words.  
10867 In K. Fiedler (Ed.), *Social communication*, pp. 343–359. New York and Hove: Psychology  
10868 Press.
- 10869 Church, K. (2011). A pendulum swung too far. *Linguistic Issues in Language Technology* 6(5),  
10870 1–27.
- 10871 Church, K. W. (2000). Empirical estimates of adaptation: the chance of two Noriega-  
10872 s is closer to  $p/2$  than  $p^2$ . In *Proceedings of the International Conference on Computational*  
10873 *Linguistics (COLING)*, pp. 180–186.
- 10874 Church, K. W. and P. Hanks (1990). Word association norms, mutual information, and  
10875 lexicography. *Computational linguistics* 16(1), 22–29.
- 10876 Ciaramita, M. and M. Johnson (2003). Supersense tagging of unknown nouns in wordnet.  
10877 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 168–  
10878 175.
- 10879 Clark, K. and C. D. Manning (2015). Entity-centric coreference resolution with model  
10880 stacking. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1405–  
10881 1415.
- 10882 Clark, K. and C. D. Manning (2016). Improving coreference resolution by learning entity-  
10883 level distributed representations. In *Proceedings of the Association for Computational Lin-*  
10884 *guistics (ACL)*.
- 10885 Clark, P. (2015). Elementary school science and math tests as a driver for ai: take the aristo  
10886 challenge! In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp.  
10887 4019–4021.

- 10888 Clarke, J., D. Goldwasser, M.-W. Chang, and D. Roth (2010). Driving semantic parsing  
10889 from the world’s response. In *Proceedings of the Conference on Natural Language Learning*  
10890 (*CoNLL*), pp. 18–27.
- 10891 Clarke, J. and M. Lapata (2008). Global inference for sentence compression: An integer  
10892 linear programming approach. *Journal of Artificial Intelligence Research* 31, 399–429.
- 10893 Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and psychologi-*  
10894 *cal measurement* 20(1), 37–46.
- 10895 Cohen, S. (2016). *Bayesian analysis in natural language processing*. Synthesis Lectures on  
10896 Human Language Technologies. San Rafael, CA: Morgan & Claypool Publishers.
- 10897 Collier, N., C. Nobata, and J.-i. Tsujii (2000). Extracting the names of genes and gene  
10898 products with a hidden markov model. In *Proceedings of the International Conference on*  
10899 *Computational Linguistics (COLING)*, pp. 201–207.
- 10900 Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *Proceed-*  
10901 *ings of the Association for Computational Linguistics (ACL)*, pp. 16–23.
- 10902 Collins, M. (2002). Discriminative training methods for hidden markov models: theory  
10903 and experiments with perceptron algorithms. In *Proceedings of Empirical Methods for*  
10904 *Natural Language Processing (EMNLP)*, pp. 1–8.
- 10905 Collins, M. (2013). Notes on natural language processing. <http://www.cs.columbia.edu/~mcollins/notes-spring2013.html>.
- 10907 Collins, M. and T. Koo (2005). Discriminative reranking for natural language parsing.  
10908 *Computational Linguistics* 31(1), 25–70.
- 10909 Collins, M. and B. Roark (2004). Incremental parsing with the perceptron algorithm. In  
10910 *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pp.  
10911 111. Association for Computational Linguistics.
- 10912 Collobert, R., K. Kavukcuoglu, and C. Farabet (2011). Torch7: A matlab-like environment  
10913 for machine learning. Technical Report EPFL-CONF-192376, EPFL.
- 10914 Collobert, R. and J. Weston (2008). A unified architecture for natural language process-  
10915 ing: Deep neural networks with multitask learning. In *Proceedings of the International*  
10916 *Conference on Machine Learning (ICML)*, pp. 160–167.
- 10917 Collobert, R., J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa (2011). Nat-  
10918 ural language processing (almost) from scratch. *Journal of Machine Learning Research* 12,  
10919 2493–2537.

- 10920 Colton, S., J. Goodwin, and T. Veale (2012). Full-face poetry generation. In *Proceedings of  
10921 the International Conference on Computational Creativity*, pp. 95–102.
- 10922 Conneau, A., D. Kiela, H. Schwenk, L. Barrault, and A. Bordes (2017). Supervised learning  
10923 of universal sentence representations from natural language inference data. In *Proceed-  
10924 ings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 681–691.
- 10925 Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2009). *Introduction to algorithms*  
10926 (third ed.). MIT press.
- 10927 Cotterell, R., H. Schütze, and J. Eisner (2016). Morphological smoothing and extrapolation  
10928 of word embeddings. In *Proceedings of the Association for Computational Linguistics (ACL)*,  
10929 pp. 1651–1660.
- 10930 Coviello, L., Y. Sohn, A. D. Kramer, C. Marlow, M. Franceschetti, N. A. Christakis, and  
10931 J. H. Fowler (2014). Detecting emotional contagion in massive social networks. *PloS  
10932 one* 9(3), e90315.
- 10933 Covington, M. A. (2001). A fundamental algorithm for dependency parsing. In *Proceedings  
10934 of the 39th annual ACM southeast conference*, pp. 95–102.
- 10935 Crammer, K., O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer (2006, December).  
10936 Online passive-aggressive algorithms. *The Journal of Machine Learning Research* 7, 551–  
10937 585.
- 10938 Crammer, K. and Y. Singer (2001). Pranking with ranking. In *Neural Information Processing  
10939 Systems (NIPS)*, pp. 641–647.
- 10940 Creutz, M. and K. Lagus (2007). Unsupervised models for morpheme segmentation and  
10941 morphology learning. *ACM Transactions on Speech and Language Processing (TSLP)* 4(1),  
10942 3.
- 10943 Cross, J. and L. Huang (2016). Span-based constituency parsing with a structure-label  
10944 system and provably optimal dynamic oracles. In *Proceedings of Empirical Methods for  
10945 Natural Language Processing (EMNLP)*, pp. 1–11.
- 10946 Cucerzan, S. (2007). Large-scale named entity disambiguation based on wikipedia data.  
10947 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 10948 Cui, H., R. Sun, K. Li, M.-Y. Kan, and T.-S. Chua (2005). Question answering passage  
10949 retrieval using dependency relations. In *Proceedings of the 28th annual international ACM  
10950 SIGIR conference on Research and development in information retrieval*, pp. 400–407. ACM.
- 10951 Cui, Y., Z. Chen, S. Wei, S. Wang, T. Liu, and G. Hu (2017). Attention-over-attention neural  
10952 networks for reading comprehension. In *Proceedings of the Association for Computational  
10953 Linguistics (ACL)*.

- 10954 Culotta, A. and J. Sorensen (2004). Dependency tree kernels for relation extraction. In  
 10955 *Proceedings of the Association for Computational Linguistics (ACL)*.
- 10956 Culotta, A., M. Wick, and A. McCallum (2007). First-order probabilistic models for coref-  
 10957 erence resolution. In *Proceedings of the North American Chapter of the Association for Com-*  
 10958 *putational Linguistics (NAACL)*, pp. 81–88.
- 10959 Curry, H. B. and R. Feys (1958). *Combinatory Logic*, Volume I. Amsterdam: North Holland.
- 10960 Danescu-Niculescu-Mizil, C., M. Sudhof, D. Jurafsky, J. Leskovec, and C. Potts (2013). A  
 10961 computational approach to politeness with application to social factors. In *Proceedings*  
 10962 *of the Association for Computational Linguistics (ACL)*, pp. 250–259.
- 10963 Das, D., D. Chen, A. F. Martins, N. Schneider, and N. A. Smith (2014). Frame-semantic  
 10964 parsing. *Computational Linguistics* 40(1), 9–56.
- 10965 Daumé III, H. (2007). Frustratingly easy domain adaptation. In *Proceedings of the Associa-*  
 10966 *tion for Computational Linguistics (ACL)*.
- 10967 Daumé III, H., J. Langford, and D. Marcu (2009). Search-based structured prediction.  
 10968 *Machine learning* 75(3), 297–325.
- 10969 Daumé III, H. and D. Marcu (2005). A large-scale exploration of effective global features  
 10970 for a joint entity detection and tracking model. In *Proceedings of Empirical Methods for*  
 10971 *Natural Language Processing (EMNLP)*, pp. 97–104.
- 10972 Dauphin, Y. N., R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio (2014). Iden-  
 10973 tifying and attacking the saddle point problem in high-dimensional non-convex opti-  
 10974 mization. In *Neural Information Processing Systems (NIPS)*, pp. 2933–2941.
- 10975 Davidson, D. (1967). The logical form of action sentences. In N. Rescher (Ed.), *The Logic of*  
 10976 *Decision and Action*. Pittsburgh: University of Pittsburgh Press.
- 10977 De Gispert, A. and J. B. Marino (2006). Catalan-english statistical machine translation  
 10978 without parallel corpus: bridging through spanish. In *Proc. of 5th International Conference*  
 10979 *on Language Resources and Evaluation (LREC)*, pp. 65–68. Citeseer.
- 10980 De Marneffe, M.-C. and C. D. Manning (2008). The stanford typed dependencies represen-  
 10981 tation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain*  
 10982 *Parser Evaluation*, pp. 1–8. Association for Computational Linguistics.
- 10983 Dean, J. and S. Ghemawat (2008). Mapreduce: simplified data processing on large clusters.  
 10984 *Communications of the ACM* 51(1), 107–113.
- 10985 Deerwester, S. C., S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman (1990).  
 10986 Indexing by latent semantic analysis. *JASIS* 41(6), 391–407.

- 10987 Dehdari, J. (2014). *A Neurophysiologically-Inspired Statistical Language Model*. Ph. D. thesis,  
10988 The Ohio State University.
- 10989 Deisenroth, M. P., A. A. Faisal, and C. S. Ong (2018). *Mathematics For Machine Learning*.  
10990 Cambridge UP.
- 10991 Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). Maximum likelihood from incom-  
10992 plete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Method-  
10993 ological)*, 1–38.
- 10994 Denis, P. and J. Baldridge (2007). A ranking approach to pronoun resolution. In *IJCAI*.
- 10995 Denis, P. and J. Baldridge (2008). Specialized models and ranking for coreference resolu-  
10996 tion. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*,  
10997 EMNLP '08, Stroudsburg, PA, USA, pp. 660–669. Association for Computational Lin-  
10998 guistics.
- 10999 Denis, P. and J. Baldridge (2009). Global joint models for coreference resolution and named  
11000 entity classification. *Procesamiento del Lenguaje Natural* 42.
- 11001 Derrida, J. (1985). Des tours de babel. In J. Graham (Ed.), *Difference in translation*. Ithaca,  
11002 NY: Cornell University Press.
- 11003 Dhingra, B., H. Liu, Z. Yang, W. W. Cohen, and R. Salakhutdinov (2017). Gated-attention  
11004 readers for text comprehension. In *Proceedings of the Association for Computational Lin-  
11005 guistics (ACL)*.
- 11006 Diaconis, P. and B. Skyrms (2017). *Ten Great Ideas About Chance*. Princeton University  
11007 Press.
- 11008 Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classifica-  
11009 tion learning algorithms. *Neural computation* 10(7), 1895–1923.
- 11010 Dietterich, T. G., R. H. Lathrop, and T. Lozano-Pérez (1997). Solving the multiple instance  
11011 problem with axis-parallel rectangles. *Artificial intelligence* 89(1), 31–71.
- 11012 Dimitrova, L., N. Ide, V. Petkevic, T. Erjavec, H. J. Kaalep, and D. Tufis (1998). Multext-  
11013 east: Parallel and comparable corpora and lexicons for six central and eastern european  
11014 languages. In *Proceedings of the 17th international conference on Computational linguistics-  
11015 Volume 1*, pp. 315–319. Association for Computational Linguistics.
- 11016 Doddington, G. R., A. Mitchell, M. A. Przybocki, L. A. Ramshaw, S. Strassel, and R. M.  
11017 Weischedel (2004). The automatic content extraction (ace) program-tasks, data, and  
11018 evaluation. In *Proceedings of the Language Resources and Evaluation Conference*, pp. 837–  
11019 840.

- 11020 dos Santos, C., B. Xiang, and B. Zhou (2015). Classifying relations by ranking with con-  
 11021     volutional neural networks. In *Proceedings of the Association for Computational Linguistics*  
 11022     (ACL), pp. 626–634.
- 11023 Dowty, D. (1991). Thematic proto-roles and argument selection. *Language*, 547–619.
- 11024 Dredze, M., P. McNamee, D. Rao, A. Gerber, and T. Finin (2010). Entity disambiguation  
 11025     for knowledge base population. In *Proceedings of the 23rd International Conference on*  
 11026     *Computational Linguistics*, pp. 277–285. Association for Computational Linguistics.
- 11027 Dredze, M., M. J. Paul, S. Bergsma, and H. Tran (2013). Carmen: A Twitter geolocation  
 11028     system with applications to public health. In *AAAI workshop on expanding the boundaries*  
 11029     *of health informatics using AI (HIAI)*, pp. 20–24.
- 11030 Dreyfus, H. L. (1992). *What computers still can't do: a critique of artificial reason*. MIT press.
- 11031 Du, L., W. Buntine, and M. Johnson (2013). Topic segmentation with a structured topic  
 11032     model. In *Proceedings of the North American Chapter of the Association for Computational*  
 11033     *Linguistics* (NAACL), pp. 190–200.
- 11034 Duchi, J., E. Hazan, and Y. Singer (2011). Adaptive subgradient methods for online learn-  
 11035     ing and stochastic optimization. *The Journal of Machine Learning Research* 12, 2121–2159.
- 11036 Dunietz, J., L. Levin, and J. Carbonell (2017). The because corpus 2.0: Annotating causality  
 11037     and overlapping relations. In *Proceedings of the Linguistic Annotation Workshop*.
- 11038 Durrett, G., T. Berg-Kirkpatrick, and D. Klein (2016). Learning-based single-document  
 11039     summarization with compression and anaphoricity constraints. In *Proceedings of the*  
 11040     *Association for Computational Linguistics* (ACL), pp. 1998–2008.
- 11041 Durrett, G. and D. Klein (2013). Easy victories and uphill battles in coreference resolution.  
 11042     In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.
- 11043 Durrett, G. and D. Klein (2015). Neural crf parsing. In *Proceedings of the Association for*  
 11044     *Computational Linguistics* (ACL).
- 11045 Dyer, C., M. Ballesteros, W. Ling, A. Matthews, and N. A. Smith (2015). Transition-based  
 11046     dependency parsing with stack long short-term memory. In *Proceedings of the Association*  
 11047     *for Computational Linguistics* (ACL), pp. 334–343.
- 11048 Dyer, C., A. Kuncoro, M. Ballesteros, and N. A. Smith (2016). Recurrent neural network  
 11049     grammars. In *Proceedings of the North American Chapter of the Association for Computational*  
 11050     *Linguistics* (NAACL), pp. 199–209.
- 11051 Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of*  
 11052     *Standards B* 71(4), 233–240.

- 11053 Efron, B. and R. J. Tibshirani (1993). An introduction to the bootstrap: Monographs on  
11054 statistics and applied probability, vol. 57. *New York and London: Chapman and Hall/CRC*.
- 11055 Eisenstein, J. (2009). Hierarchical text segmentation from multi-scale lexical cohesion. In  
11056 *Proceedings of the North American Chapter of the Association for Computational Linguistics*  
11057 (NAACL).
- 11058 Eisenstein, J. and R. Barzilay (2008). Bayesian unsupervised topic segmentation. In *Pro-*  
11059 *ceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11060 Eisner, J. (1997). State-of-the-art algorithms for minimum spanning trees: A tutorial dis-  
11061 cussion.
- 11062 Eisner, J. (2000). Bilexical grammars and their cubic-time parsing algorithms. In *Advances*  
11063 *in probabilistic and other parsing technologies*, pp. 29–61. Springer.
- 11064 Eisner, J. (2002). Parameter estimation for probabilistic finite-state transducers. In *Proced-*  
11065 *ings of the Association for Computational Linguistics (ACL)*, pp. 1–8.
- 11066 Eisner, J. (2016). Inside-outside and forward-backward algorithms are just backprop. In  
11067 *Proceedings of the Workshop on Structured Prediction for NLP*, pp. 1–17.
- 11068 Eisner, J. M. (1996). Three new probabilistic models for dependency parsing: An explo-  
11069 ration. In *Proceedings of the International Conference on Computational Linguistics (COL-*  
11070 *ING)*, pp. 340–345.
- 11071 Ekman, P. (1992). Are there basic emotions? *Psychological Review* 99(3), 550–553.
- 11072 Elman, J. L. (1990). Finding structure in time. *Cognitive science* 14(2), 179–211.
- 11073 Elman, J. L., E. A. Bates, M. H. Johnson, A. Karmiloff-Smith, D. Parisi, and K. Plunkett  
11074 (1998). *Rethinking innateness: A connectionist perspective on development*, Volume 10. MIT  
11075 press.
- 11076 Elsner, M. and E. Charniak (2010). Disentangling chat. *Computational Linguistics* 36(3),  
11077 389–409.
- 11078 Esuli, A. and F. Sebastiani (2006). Sentiwordnet: A publicly available lexical resource for  
11079 opinion mining. In *LREC*, Volume 6, pp. 417–422. Citeseer.
- 11080 Etzioni, O., A. Fader, J. Christensen, S. Soderland, and M. Mausam (2011). Open informa-  
11081 tion extraction: The second generation. In *Proceedings of the International Joint Conference*  
11082 *on Artificial Intelligence (IJCAI)*, pp. 3–10.

- 11083 Faruqui, M., J. Dodge, S. K. Jauhar, C. Dyer, E. Hovy, and N. A. Smith (2015). Retrofitting  
11084 word vectors to semantic lexicons. In *Proceedings of the North American Chapter of the*  
11085 *Association for Computational Linguistics (NAACL)*.
- 11086 Faruqui, M. and C. Dyer (2014). Improving vector space word representations using mul-  
11087 tilingual correlation. In *Proceedings of the European Chapter of the Association for Compu-*  
11088 *tational Linguistics (EACL)*, pp. 462–471.
- 11089 Faruqui, M., R. McDonald, and R. Soricut (2016). Morpho-syntactic lexicon generation  
11090 using graph-based semi-supervised learning. *Transactions of the Association for Compu-*  
11091 *tational Linguistics* 4, 1–16.
- 11092 Faruqui, M., Y. Tsvetkov, P. Rastogi, and C. Dyer (2016, August). Problems with evaluation  
11093 of word embeddings using word similarity tasks. In *Proceedings of the 1st Workshop on*  
11094 *Evaluating Vector-Space Representations for NLP*, Berlin, Germany, pp. 30–35. Association  
11095 for Computational Linguistics.
- 11096 Fellbaum, C. (2010). *WordNet*. Springer.
- 11097 Feng, V. W., Z. Lin, and G. Hirst (2014). The impact of deep hierarchical discourse struc-  
11098 tures in the evaluation of text coherence. In *Proceedings of the International Conference on*  
11099 *Computational Linguistics (COLING)*, pp. 940–949.
- 11100 Feng, X., L. Huang, D. Tang, H. Ji, B. Qin, and T. Liu (2016). A language-independent  
11101 neural network for event detection. In *Proceedings of the Association for Computational*  
11102 *Linguistics (ACL)*, pp. 66–71.
- 11103 Fernandes, E. R., C. N. dos Santos, and R. L. Milidiú (2014). Latent trees for coreference  
11104 resolution. *Computational Linguistics*.
- 11105 Ferrucci, D., E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W.  
11106 Murdock, E. Nyberg, J. Prager, et al. (2010). Building Watson: An overview of the  
11107 DeepQA project. *AI magazine* 31(3), 59–79.
- 11108 Ficler, J. and Y. Goldberg (2017, September). Controlling linguistic style aspects in neural  
11109 language generation. In *Proceedings of the Workshop on Stylistic Variation*, Copenhagen,  
11110 Denmark, pp. 94–104. Association for Computational Linguistics.
- 11111 Filippova, K. and M. Strube (2008). Sentence fusion via dependency graph compression.  
11112 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 177–  
11113 185.
- 11114 Fillmore, C. J. (1968). The case for case. In E. Bach and R. Harms (Eds.), *Universals in*  
11115 *linguistic theory*. Holt, Rinehart, and Winston.

- 11116 Fillmore, C. J. (1976). Frame semantics and the nature of language. *Annals of the New York  
11117 Academy of Sciences* 280(1), 20–32.
- 11118 Fillmore, C. J. and C. Baker (2009). A frames approach to semantic analysis. In *The Oxford  
11119 Handbook of Linguistic Analysis*. Oxford University Press.
- 11120 Finkel, J. R., T. Grenager, and C. Manning (2005). Incorporating non-local information  
11121 into information extraction systems by gibbs sampling. In *Proceedings of the Association  
11122 for Computational Linguistics (ACL)*, pp. 363–370.
- 11123 Finkel, J. R., T. Grenager, and C. D. Manning (2007). The infinite tree. In *Proceedings of the  
11124 Association for Computational Linguistics (ACL)*, pp. 272–279.
- 11125 Finkel, J. R., A. Kleeman, and C. D. Manning (2008). Efficient, feature-based, conditional  
11126 random field parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*,  
11127 pp. 959–967.
- 11128 Finkel, J. R. and C. Manning (2009). Hierarchical bayesian domain adaptation. In *Proceed-  
11129 ings of the North American Chapter of the Association for Computational Linguistics (NAACL)*,  
11130 pp. 602–610.
- 11131 Finkel, J. R. and C. D. Manning (2008). Enforcing transitivity in coreference resolution.  
11132 In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics  
11133 on Human Language Technologies: Short Papers*, pp. 45–48. Association for Computational  
11134 Linguistics.
- 11135 Finkelstein, L., E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin  
11136 (2002). Placing search in context: The concept revisited. *ACM Transactions on Information  
11137 Systems* 20(1), 116–131.
- 11138 Firth, J. R. (1957). *Papers in Linguistics 1934-1951*. Oxford University Press.
- 11139 Flanigan, J., S. Thomson, J. Carbonell, C. Dyer, and N. A. Smith (2014). A discrimina-  
11140 tive graph-based parser for the abstract meaning representation. In *Proceedings of the  
11141 Association for Computational Linguistics (ACL)*, pp. 1426–1436.
- 11142 Foltz, P. W., W. Kintsch, and T. K. Landauer (1998). The measurement of textual coherence  
11143 with latent semantic analysis. *Discourse processes* 25(2-3), 285–307.
- 11144 Fordyce, C. S. (2007). Overview of the iwslt 2007 evaluation campaign. In *International  
11145 Workshop on Spoken Language Translation (IWSLT) 2007*.
- 11146 Fox, H. (2002). Phrasal cohesion and statistical machine translation. In *Proceedings of  
11147 Empirical Methods for Natural Language Processing (EMNLP)*, pp. 304–3111.

- 11148 Francis, W. and H. Kucera (1982). *Frequency analysis of English usage*. Houghton Mifflin  
 11149 Company.
- 11150 Francis, W. N. (1964). A standard sample of present-day English for use with digital  
 11151 computers. Report to the U.S Office of Education on Cooperative Research Project No.  
 11152 E-007.
- 11153 Freund, Y. and R. E. Schapire (1999). Large margin classification using the perceptron  
 11154 algorithm. *Machine learning* 37(3), 277–296.
- 11155 Fromkin, V., R. Rodman, and N. Hyams (2013). *An introduction to language*. Cengage  
 11156 Learning.
- 11157 Fundel, K., R. Küffner, and R. Zimmer (2007). Relex – relation extraction using depen-  
 11158 dency parse trees. *Bioinformatics* 23(3), 365–371.
- 11159 Gabow, H. N., Z. Galil, T. Spencer, and R. E. Tarjan (1986). Efficient algorithms for finding  
 11160 minimum spanning trees in undirected and directed graphs. *Combinatorica* 6(2), 109–  
 11161 122.
- 11162 Gabrilovich, E. and S. Markovitch (2007). Computing semantic relatedness using  
 11163 wikipedia-based explicit semantic analysis. In *Proceedings of the International Joint Con-  
 11164 ference on Artificial Intelligence (IJCAI)*, Volume 7, pp. 1606–1611.
- 11165 Gage, P. (1994). A new algorithm for data compression. *The C Users Journal* 12(2), 23–38.
- 11166 Gale, W. A., K. W. Church, and D. Yarowsky (1992). One sense per discourse. In *Pro-  
 11167 ceedings of the workshop on Speech and Natural Language*, pp. 233–237. Association for  
 11168 Computational Linguistics.
- 11169 Galley, M., M. Hopkins, K. Knight, and D. Marcu (2004). What's in a translation rule? In  
 11170 *Proceedings of the North American Chapter of the Association for Computational Linguistics  
 11171 (NAACL)*, pp. 273–280.
- 11172 Galley, M., K. R. McKeown, E. Fosler-Lussier, and H. Jing (2003). Discourse segmentation  
 11173 of multi-party conversation. In *Proceedings of the Association for Computational Linguistics  
 11174 (ACL)*.
- 11175 Ganchev, K. and M. Dredze (2008). Small statistical models by random feature mixing. In  
 11176 *Proceedings of the ACL08 HLT Workshop on Mobile Language Processing*, pp. 19–20.
- 11177 Ganchev, K., J. Graça, J. Gillenwater, and B. Taskar (2010). Posterior regularization for  
 11178 structured latent variable models. *The Journal of Machine Learning Research* 11, 2001–  
 11179 2049.

- 11180 Ganin, Y., E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand,  
11181 and V. Lempitsky (2016). Domain-adversarial training of neural networks. *Journal of  
11182 Machine Learning Research* 17(59), 1–35.
- 11183 Gao, J., G. Andrew, M. Johnson, and K. Toutanova (2007). A comparative study of param-  
11184 eter estimation methods for statistical natural language processing. In *Proceedings of the  
11185 Association for Computational Linguistics (ACL)*, pp. 824–831.
- 11186 Gatt, A. and E. Krahmer (2018). Survey of the state of the art in natural language genera-  
11187 tion: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research* 61,  
11188 65–170.
- 11189 Gatt, A. and E. Reiter (2009). Simplenlg: A realisation engine for practical applications.  
11190 In *Proceedings of the 12th European Workshop on Natural Language Generation*, pp. 90–93.  
11191 Association for Computational Linguistics.
- 11192 Ge, D., X. Jiang, and Y. Ye (2011). A note on the complexity of  $l_1 p$  minimization. *Mathe-  
11193 matical programming* 129(2), 285–299.
- 11194 Ge, N., J. Hale, and E. Charniak (1998). A statistical approach to anaphora resolution. In  
11195 *Proceedings of the sixth workshop on very large corpora*, Volume 71, pp. 76.
- 11196 Ge, R., F. Huang, C. Jin, and Y. Yuan (2015). Escaping from saddle points — online stochas-  
11197 tic gradient for tensor decomposition. In P. Grünwald, E. Hazan, and S. Kale (Eds.),  
11198 *Proceedings of the Conference On Learning Theory (COLT)*.
- 11199 Ge, R. and R. J. Mooney (2005). A statistical semantic parser that integrates syntax and  
11200 semantics. In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp.  
11201 9–16.
- 11202 Geach, P. T. (1962). *Reference and generality: An examination of some medieval and modern  
11203 theories*. Cornell University Press.
- 11204 Gildea, D. and D. Jurafsky (2002). Automatic labeling of semantic roles. *Computational  
11205 linguistics* 28(3), 245–288.
- 11206 Gimpel, K., N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yo-  
11207 gatama, J. Flanigan, and N. A. Smith (2011). Part-of-speech tagging for Twitter: an-  
11208 notation, features, and experiments. In *Proceedings of the Association for Computational  
11209 Linguistics (ACL)*, pp. 42–47.
- 11210 Glass, J., T. J. Hazen, S. Cyphers, I. Malioutov, D. Huynh, and R. Barzilay (2007). Recent  
11211 progress in the mit spoken lecture processing project. In *Eighth Annual Conference of the  
11212 International Speech Communication Association*.

- 11213 Glorot, X. and Y. Bengio (2010). Understanding the difficulty of training deep feedforward  
 11214 neural networks. In *Proceedings of Artificial Intelligence and Statistics (AISTATS)*, pp. 249–  
 11215 256.
- 11216 Glorot, X., A. Bordes, and Y. Bengio (2011). Deep sparse rectifier networks. In *Proceedings*  
 11217 *of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP*  
 11218 *Volume*, Volume 15, pp. 315–323.
- 11219 Godfrey, J. J., E. C. Holliman, and J. McDaniel (1992). Switchboard: Telephone speech  
 11220 corpus for research and development. In *Proceedings of the International Conference on*  
 11221 *Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 517–520. IEEE.
- 11222 Goldberg, Y. (2017a, June). An adversarial review of “adversarial generation of  
 11223 natural language”. [https://medium.com/@yoav.goldberg/an-adversarial-review-of-](https://medium.com/@yoav.goldberg/an-adversarial-review-of-adversarial-generation-of-natural-language-409ac3378bd7)  
 11224 [adversarial-generation-of-natural-language-409ac3378bd7](https://medium.com/@yoav.goldberg/an-adversarial-review-of-natural-language-409ac3378bd7).
- 11225 Goldberg, Y. (2017b). *Neural Network Methods for Natural Language Processing*. Synthesis  
 11226 Lectures on Human Language Technologies. Morgan & Claypool Publishers.
- 11227 Goldberg, Y. and M. Elhadad (2010). An efficient algorithm for easy-first non-directional  
 11228 dependency parsing. In *Proceedings of the North American Chapter of the Association for*  
 11229 *Computational Linguistics (NAACL)*, pp. 742–750.
- 11230 Goldberg, Y. and J. Nivre (2012). A dynamic oracle for arc-eager dependency parsing.  
 11231 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.  
 11232 959–976.
- 11233 Goldberg, Y., K. Zhao, and L. Huang (2013). Efficient implementation of beam-search  
 11234 incremental parsers. In *ACL (2)*, pp. 628–633.
- 11235 Goldwater, S. and T. Griffiths (2007). A fully bayesian approach to unsupervised part-of-  
 11236 speech tagging. In *Annual meeting-association for computational linguistics*, Volume 45.
- 11237 Gonçalo Oliveira, H. R., F. A. Cardoso, and F. C. Pereira (2007). Tra-la-lyrics: An approach  
 11238 to generate text based on rhythm. In *Proceedings of the 4th. International Joint Workshop*  
 11239 *on Computational Creativity*. A. Cardoso and G. Wiggins.
- 11240 Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep learning*. MIT Press.
- 11241 Goodman, J. T. (2001). A bit of progress in language modeling. *Computer Speech & Lan-*  
 11242 *guage* 15(4), 403–434.
- 11243 Gouws, S., D. Metzler, C. Cai, and E. Hovy (2011). Contextual bearing on linguistic varia-  
 11244 *tion in social media*. In *LASM*.

- 11245 Goyal, A., H. Daume III, and S. Venkatasubramanian (2009). Streaming for large scale  
11246 nlp: Language modeling. In *Proceedings of the North American Chapter of the Association*  
11247 for Computational Linguistics (NAACL), pp. 512–520.
- 11248 Graves, A. (2012). Sequence transduction with recurrent neural networks. In *Proceedings*  
11249 of the International Conference on Machine Learning (ICML).
- 11250 Graves, A. and N. Jaitly (2014). Towards end-to-end speech recognition with recur-  
11251 rent neural networks. In *Proceedings of the International Conference on Machine Learning*  
11252 (ICML), pp. 1764–1772.
- 11253 Graves, A. and J. Schmidhuber (2005). Framewise phoneme classification with bidirec-  
11254 tional lstm and other neural network architectures. *Neural Networks* 18(5), 602–610.
- 11255 Grice, H. P. (1975). Logic and conversation. In P. Cole and J. L. Morgan (Eds.), *Syntax and*  
11256 *Semantics Volume 3: Speech Acts*, pp. 41–58. Academic Press.
- 11257 Grishman, R. (2012). Information extraction: Capabilities and challenges. Notes prepared  
11258 for the 2012 International Winter School in Language and Speech Technologies, Rovira  
11259 i Virgili University, Tarragona, Spain.
- 11260 Grishman, R. (2015). Information extraction. *IEEE Intelligent Systems* 30(5), 8–15.
- 11261 Grishman, R., C. Macleod, and J. Sterling (1992). Evaluating parsing strategies using  
11262 standardized parse files. In *Proceedings of the third conference on Applied natural language*  
11263 *processing*, pp. 156–161. Association for Computational Linguistics.
- 11264 Grishman, R. and B. Sundheim (1996). Message understanding conference-6: A brief his-  
11265 tory. In *Proceedings of the International Conference on Computational Linguistics* (COLING),  
11266 pp. 466–471.
- 11267 Groenendijk, J. and M. Stokhof (1991). Dynamic predicate logic. *Linguistics and philoso-*  
11268 *phy* 14(1), 39–100.
- 11269 Grosz, B. J. (1979). Focusing and description in natural language dialogues. Technical  
11270 report, SRI INTERNATIONAL MENLO PARK CA.
- 11271 Grosz, B. J., S. Weinstein, and A. K. Joshi (1995). Centering: A framework for modeling  
11272 the local coherence of discourse. *Computational linguistics* 21(2), 203–225.
- 11273 Gu, J., Z. Lu, H. Li, and V. O. Li (2016). Incorporating copying mechanism in sequence-to-  
11274 sequence learning. In *Proceedings of the Association for Computational Linguistics* (ACL),  
11275 pp. 1631–1640.
- 11276 Gulcehre, C., S. Ahn, R. Nallapati, B. Zhou, and Y. Bengio (2016). Pointing the unknown  
11277 words. In *Proceedings of the Association for Computational Linguistics* (ACL), pp. 140–149.

- 11278 Gutmann, M. U. and A. Hyvärinen (2012). Noise-contrastive estimation of unnormalized  
11279 statistical models, with applications to natural image statistics. *The Journal of Machine  
11280 Learning Research* 13(1), 307–361.
- 11281 Haghghi, A. and D. Klein (2007). Unsupervised coreference resolution in a nonparametric  
11282 bayesian model. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11283 Haghghi, A. and D. Klein (2009). Simple coreference resolution with rich syntactic and  
11284 semantic features. In *Proceedings of Empirical Methods for Natural Language Processing  
(EMNLP)*, pp. 1152–1161.
- 11286 Haghghi, A. and D. Klein (2010). Coreference resolution in a modular, entity-centered  
11287 model. In *Proceedings of the North American Chapter of the Association for Computational  
11288 Linguistics (NAACL)*, pp. 385–393.
- 11289 Hajič, J. and B. Hladká (1998). Tagging inflective languages: Prediction of morphological  
11290 categories for a rich, structured tagset. In *Proceedings of the Association for Computational  
11291 Linguistics (ACL)*, pp. 483–490.
- 11292 Halliday, M. and R. Hasan (1976). *Cohesion in English*. London: Longman.
- 11293 Hammerton, J. (2003). Named entity recognition with long short-term memory. In *Pro-  
11294 ceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 172–175.
- 11295 Han, X. and L. Sun (2012). An entity-topic model for entity linking. In *Proceedings of  
11296 Empirical Methods for Natural Language Processing (EMNLP)*, pp. 105–115.
- 11297 Han, X., L. Sun, and J. Zhao (2011). Collective entity linking in web text: a graph-based  
11298 method. In *Proceedings of ACM SIGIR conference on Research and development in informa-  
11299 tion retrieval*, pp. 765–774.
- 11300 Hannak, A., E. Anderson, L. F. Barrett, S. Lehmann, A. Mislove, and M. Riedewald (2012).  
11301 Tweetin'in the rain: Exploring societal-scale effects of weather on mood. In *Proceedings  
11302 of the International Conference on Web and Social Media (ICWSM)*.
- 11303 Hardmeier, C. (2012). Discourse in statistical machine translation. a survey and a case  
11304 study. *Discours. Revue de linguistique, psycholinguistique et informatique. A journal of lin-  
11305 guistics, psycholinguistics and computational linguistics* (11).
- 11306 Haspelmath, M. and A. Sims (2013). *Understanding morphology*. Routledge.
- 11307 Hastie, T., R. Tibshirani, and J. Friedman (2009). *The elements of statistical learning* (Second  
11308 ed.). New York: Springer.

- 11309 Hatzivassiloglou, V. and K. R. McKeown (1997). Predicting the semantic orientation of  
11310 adjectives. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 174–  
11311 181.
- 11312 Hayes, A. F. and K. Krippendorff (2007). Answering the call for a standard reliability  
11313 measure for coding data. *Communication methods and measures* 1(1), 77–89.
- 11314 He, H., A. Balakrishnan, M. Eric, and P. Liang (2017). Learning symmetric collaborative  
11315 dialogue agents with dynamic knowledge graph embeddings. In *Proceedings of the As-  
11316 sociation for Computational Linguistics (ACL)*, pp. 1766–1776.
- 11317 He, K., X. Zhang, S. Ren, and J. Sun (2015). Delving deep into rectifiers: Surpassing  
11318 human-level performance on imagenet classification. In *Proceedings of the International  
11319 Conference on Computer Vision (ICCV)*, pp. 1026–1034.
- 11320 He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep residual learning for image recognition.  
11321 In *Proceedings of the International Conference on Computer Vision (ICCV)*, pp. 770–778.
- 11322 He, L., K. Lee, M. Lewis, and L. Zettlemoyer (2017). Deep semantic role labeling: What  
11323 works and what's next. In *Proceedings of the Association for Computational Linguistics  
11324 (ACL)*.
- 11325 He, Z., S. Liu, M. Li, M. Zhou, L. Zhang, and H. Wang (2013). Learning entity repre-  
11326 sentation for entity disambiguation. In *Proceedings of the Association for Computational  
11327 Linguistics (ACL)*, pp. 30–34.
- 11328 Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. In  
11329 *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp. 539–  
11330 545. Association for Computational Linguistics.
- 11331 Hearst, M. A. (1997). Texttiling: Segmenting text into multi-paragraph subtopic passages.  
11332 *Computational linguistics* 23(1), 33–64.
- 11333 Heerschap, B., F. Goossen, A. Hogenboom, F. Frasincar, U. Kaymak, and F. de Jong (2011).  
11334 Polarity analysis of texts using discourse structure. In *Proceedings of the 20th ACM inter-  
11335 national conference on Information and knowledge management*, pp. 1061–1070. ACM.
- 11336 Henderson, J. (2004). Discriminative training of a neural network statistical parser. In  
11337 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 95–102.
- 11338 Hendrickx, I., S. N. Kim, Z. Kozareva, P. Nakov, D. Ó Séaghdha, S. Padó, M. Pennacchiotti,  
11339 L. Romano, and S. Szpakowicz (2009). SemEval-2010 task 8: Multi-way classification of  
11340 semantic relations between pairs of nominals. In *Proceedings of the Workshop on Semantic  
11341 Evaluations: Recent Achievements and Future Directions*, pp. 94–99. Association for Com-  
11342 putational Linguistics.

- 11343 Hermann, K. M., T. Kočiský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and  
 11344 P. Blunsom (2015). Teaching machines to read and comprehend. In *Advances in Neu-*  
 11345 *ral Information Processing Systems*, pp. 1693–1701.
- 11346 Hernault, H., H. Prendinger, D. A. duVerle, and M. Ishizuka (2010). HILDA: A discourse  
 11347 parser using support vector machine classification. *Dialogue and Discourse* 1(3), 1–33.
- 11348 Hill, F., A. Bordes, S. Chopra, and J. Weston (2016). The goldilocks principle: Reading  
 11349 children’s books with explicit memory representations. In *Proceedings of the International*  
 11350 *Conference on Learning Representations (ICLR)*.
- 11351 Hill, F., K. Cho, and A. Korhonen (2016). Learning distributed representations of sentences  
 11352 from unlabelled data. In *Proceedings of the North American Chapter of the Association for*  
 11353 *Computational Linguistics (NAACL)*.
- 11354 Hindle, D. and M. Rooth (1993). Structural ambiguity and lexical relations. *Computational*  
 11355 *linguistics* 19(1), 103–120.
- 11356 Hirao, T., Y. Yoshida, M. Nishino, N. Yasuda, and M. Nagata (2013). Single-document  
 11357 summarization as a tree knapsack problem. In *Proceedings of Empirical Methods for Nat-*  
 11358 *ural Language Processing (EMNLP)*, pp. 1515–1520.
- 11359 Hirschman, L. and R. Gaizauskas (2001). Natural language question answering: the view  
 11360 from here. *natural language engineering* 7(4), 275–300.
- 11361 Hirschman, L., M. Light, E. Breck, and J. D. Burger (1999). Deep read: A reading compre-  
 11362 hension system. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.  
 11363 325–332.
- 11364 Hobbs, J. R. (1978). Resolving pronoun references. *Lingua* 44(4), 311–338.
- 11365 Hobbs, J. R., D. Appelt, J. Bear, D. Israel, M. Kameyama, M. Stickel, and M. Tyson (1997).  
 11366 Fastus: A cascaded finite-state transducer for extracting information from natural-  
 11367 language text. *Finite-state language processing*, 383–406.
- 11368 Hochreiter, S. and J. Schmidhuber (1997). Long short-term memory. *Neural computa-*  
 11369 *tion* 9(8), 1735–1780.
- 11370 Hockenmaier, J. and M. Steedman (2007). Ccgbank: a corpus of ccg derivations and de-  
 11371 pendency structures extracted from the penn treebank. *Computational Linguistics* 33(3),  
 11372 355–396.
- 11373 Hoffart, J., M. A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, B. Taneva,  
 11374 S. Thater, and G. Weikum (2011). Robust disambiguation of named entities in text. In  
 11375 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 782–792.

- 11376 Hoffmann, R., C. Zhang, X. Ling, L. Zettlemoyer, and D. S. Weld (2011). Knowledge-based  
11377 weak supervision for information extraction of overlapping relations. In *Proceedings of*  
11378 *the Association for Computational Linguistics (ACL)*, pp. 541–550.
- 11379 Holmstrom, L. and P. Koistinen (1992). Using additive noise in back-propagation training.  
11380 *IEEE Transactions on Neural Networks* 3(1), 24–38.
- 11381 Hovy, E. and J. Lavid (2010). Towards a ‘science’ of corpus annotation: a new method-  
11382 ological challenge for corpus linguistics. *International journal of translation* 22(1), 13–36.
- 11383 Hsu, D., S. M. Kakade, and T. Zhang (2012). A spectral algorithm for learning hidden  
11384 markov models. *Journal of Computer and System Sciences* 78(5), 1460–1480.
- 11385 Hu, M. and B. Liu (2004). Mining and summarizing customer reviews. In *Proceedings of*  
11386 *Knowledge Discovery and Data Mining (KDD)*, pp. 168–177.
- 11387 Hu, Z., Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing (2017). Toward controlled  
11388 generation of text. In *International Conference on Machine Learning*, pp. 1587–1596.
- 11389 Huang, F. and A. Yates (2012). Biased representation learning for domain adaptation. In  
11390 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1313–1323.
- 11391 Huang, L. and D. Chiang (2007). Forest rescoring: Faster decoding with integrated lan-  
11392 guage models. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.  
11393 144–151.
- 11394 Huang, L., S. Fayong, and Y. Guo (2012). Structured perceptron with inexact search. In  
11395 *Proceedings of the North American Chapter of the Association for Computational Linguistics*  
11396 (*NAACL*), pp. 142–151.
- 11397 Huang, Y. (2015). *Pragmatics* (Second ed.). Oxford Textbooks in Linguistics. Oxford Uni-  
11398 versity Press.
- 11399 Huang, Z., W. Xu, and K. Yu (2015). Bidirectional lstm-crf models for sequence tagging.  
11400 *arXiv preprint arXiv:1508.01991*.
- 11401 Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes.  
11402 *Proceedings of the IRE* 40(9), 1098–1101.
- 11403 Humphreys, K., R. Gaizauskas, and S. Azzam (1997). Event coreference for information  
11404 extraction. In *Proceedings of a Workshop on Operational Factors in Practical, Robust Anaphora*  
11405 *Resolution for Unrestricted Texts*, pp. 75–81. Association for Computational Linguistics.
- 11406 Ide, N. and Y. Wilks (2006). Making sense about sense. In *Word sense disambiguation*, pp.  
11407 47–73. Springer.

- 11408 Ioffe, S. and C. Szegedy (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 448–456.
- 11411 Isozaki, H., T. Hirao, K. Duh, K. Sudoh, and H. Tsukada (2010). Automatic evaluation of translation quality for distant language pairs. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 944–952.
- 11414 Iyyer, M., V. Manjunatha, J. Boyd-Graber, and H. Daumé III (2015). Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1681–1691.
- 11417 James, G., D. Witten, T. Hastie, and R. Tibshirani (2013). *An introduction to statistical learning*, Volume 112. Springer.
- 11419 Janin, A., D. Baron, J. Edwards, D. Ellis, D. Gelbart, N. Morgan, B. Peskin, T. Pfau, E. Shriberg, A. Stolcke, et al. (2003). The ICSI meeting corpus. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, Volume 1, pp. I–I. IEEE.
- 11423 Jean, S., K. Cho, R. Memisevic, and Y. Bengio (2015). On using very large target vocabulary for neural machine translation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1–10.
- 11426 Jeong, M., C.-Y. Lin, and G. G. Lee (2009). Semi-supervised speech act recognition in emails and forums. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1250–1259.
- 11429 Ji, H. and R. Grishman (2011). Knowledge base population: Successful approaches and challenges. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1148–1158.
- 11432 Ji, Y., T. Cohn, L. Kong, C. Dyer, and J. Eisenstein (2015). Document context language models. In *International Conference on Learning Representations, Workshop Track*, Volume abs/1511.03962.
- 11435 Ji, Y. and J. Eisenstein (2014). Representation learning for text-level discourse parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11437 Ji, Y. and J. Eisenstein (2015, June). One vector is not enough: Entity-augmented distributional semantics for discourse relations. *Transactions of the Association for Computational Linguistics (TACL)*.

- 11440 Ji, Y., G. Haffari, and J. Eisenstein (2016). A latent variable recurrent neural network for  
11441 discourse relation language models. In *Proceedings of the North American Chapter of the*  
11442 *Association for Computational Linguistics (NAACL)*.
- 11443 Ji, Y. and N. A. Smith (2017). Neural discourse structure for text categorization. In *Pro-  
11444 ceedings of the Association for Computational Linguistics (ACL)*, pp. 996–1005.
- 11445 Ji, Y., C. Tan, S. Martschat, Y. Choi, and N. A. Smith (2017). Dynamic entity representations  
11446 in neural language models. In *Proceedings of Empirical Methods for Natural Language  
11447 Processing (EMNLP)*, pp. 1831–1840.
- 11448 Jiang, L., M. Yu, M. Zhou, X. Liu, and T. Zhao (2011). Target-dependent twitter sentiment  
11449 classification. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.  
11450 151–160.
- 11451 Jing, H. (2000). Sentence reduction for automatic text summarization. In *Proceedings of  
the sixth conference on Applied natural language processing*, pp. 310–315. Association for  
11452 Computational Linguistics.
- 11453 Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of  
Knowledge Discovery and Data Mining (KDD)*, pp. 133–142.
- 11454 Jockers, M. L. (2015). Szuzhet? <http://bla.bla.com>.
- 11455 Johnson, A. E., T. J. Pollard, L. Shen, H. L. Li-wei, M. Feng, M. Ghassemi, B. Moody,  
11456 P. Szolovits, L. A. Celi, and R. G. Mark (2016). Mimic-iii, a freely accessible critical care  
11457 database. *Scientific data* 3, 160035.
- 11458 Johnson, M. (1998). Pcfg models of linguistic tree representations. *Computational Linguis-  
11459 tics* 24(4), 613–632.
- 11460 Johnson, R. and T. Zhang (2017). Deep pyramid convolutional neural networks for text  
11461 categorization. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.  
11462 562–570.
- 11463 Joshi, A. K. (1985). How much context-sensitivity is required to provide reasonable struc-  
11464 tural descriptions? – tree adjoining grammars. In *Natural Language Processing – Theoret-  
11465 ical, Computational and Psychological Perspective*. New York, NY: Cambridge University  
11466 Press.
- 11467 Joshi, A. K. and Y. Schabes (1997). Tree-adjoining grammars. In *Handbook of formal lan-  
11468 guages*, pp. 69–123. Springer.
- 11469 Joshi, A. K., K. V. Shanker, and D. Weir (1991). The convergence of mildly context-sensitive  
11470 grammar formalisms. In *Foundational Issues in Natural Language Processing*. Cambridge  
11471 MA: MIT Press.

- 11474 Jozefowicz, R., O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu (2016). Exploring the limits  
11475 of language modeling. *arXiv preprint arXiv:1602.02410*.
- 11476 Jozefowicz, R., W. Zaremba, and I. Sutskever (2015). An empirical exploration of recurrent  
11477 network architectures. In *Proceedings of the International Conference on Machine Learning*  
11478 (*ICML*), pp. 2342–2350.
- 11479 Jurafsky, D. (1996). A probabilistic model of lexical and syntactic access and disambiguation.  
11480 *Cognitive Science* 20(2), 137–194.
- 11481 Jurafsky, D. and J. H. Martin (2009). *Speech and Language Processing* (Second ed.). Prentice  
11482 Hall.
- 11483 Jurafsky, D. and J. H. Martin (2018). *Speech and Language Processing* (Third ed.). Prentice  
11484 Hall.
- 11485 Kadlec, R., M. Schmid, O. Bajgar, and J. Kleindienst (2016). Text understanding with  
11486 the attention sum reader network. In *Proceedings of the Association for Computational*  
11487 *Linguistics (ACL)*, pp. 908–918.
- 11488 Kalchbrenner, N. and P. Blunsom (2013, August). Recurrent convolutional neural net-  
11489 works for discourse compositionality. In *Proceedings of the Workshop on Continuous Vec-*  
11490 *tor Space Models and their Compositionality*, Sofia, Bulgaria, pp. 119–126. Association for  
11491 Computational Linguistics.
- 11492 Kalchbrenner, N., E. Grefenstette, and P. Blunsom (2014). A convolutional neural network  
11493 for modelling sentences. In *Proceedings of the Association for Computational Linguistics*  
11494 (*ACL*), pp. 655–665.
- 11495 Karlsson, F. (2007). Constraints on multiple center-embedding of clauses. *Journal of Lin-*  
11496 *guistics* 43(02), 365–392.
- 11497 Kate, R. J., Y. W. Wong, and R. J. Mooney (2005). Learning to transform natural to formal  
11498 languages. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.
- 11499 Kehler, A. (2007). Rethinking the SMASH approach to pronoun interpretation. In *Interdis-*  
11500 *ciplinary perspectives on reference processing*, New Directions in Cognitive Science Series,  
11501 pp. 95–122. Oxford University Press.
- 11502 Kibble, R. and R. Power (2004). Optimizing referential coherence in text generation. *Com-*  
11503 *putational Linguistics* 30(4), 401–416.
- 11504 Kilgarriff, A. (1997). I don't believe in word senses. *Computers and the Humanities* 31(2),  
11505 91–113.

- 11506 Kilgarriff, A. and G. Grefenstette (2003). Introduction to the special issue on the web as  
11507 corpus. *Computational linguistics* 29(3), 333–347.
- 11508 Kim, M.-J. (2002). Does korean have adjectives? *MIT Working Papers in Linguistics* 43,  
11509 71–89.
- 11510 Kim, S.-M. and E. Hovy (2006, July). Extracting opinions, opinion holders, and topics  
11511 expressed in online news media text. In *Proceedings of the Workshop on Sentiment and*  
11512 *Subjectivity in Text*, Sydney, Australia, pp. 1–8. Association for Computational Linguis-  
11513 tics.
- 11514 Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings*  
11515 *of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1746–1751.
- 11516 Kim, Y., C. Denton, L. Hoang, and A. M. Rush (2017). Structured attention networks. In  
11517 *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 11518 Kim, Y., Y. Jernite, D. Sontag, and A. M. Rush (2016). Character-aware neural language  
11519 models. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.
- 11520 Kingma, D. and J. Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint*  
11521 *arXiv:1412.6980*.
- 11522 Kiperwasser, E. and Y. Goldberg (2016). Simple and accurate dependency parsing using  
11523 bidirectional lstm feature representations. *Transactions of the Association for Compu-  
11524 tational Linguistics* 4, 313–327.
- 11525 Kipper-Schuler, K. (2005). *VerbNet: A broad-coverage, comprehensive verb lexicon*. Ph. D.  
11526 thesis, Computer and Information Science, University of Pennsylvania.
- 11527 Kiros, R., R. Salakhutdinov, and R. Zemel (2014). Multimodal neural language models. In  
11528 *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 595–603.
- 11529 Kiros, R., Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler  
11530 (2015). Skip-thought vectors. In *Neural Information Processing Systems (NIPS)*.
- 11531 Klein, D. and C. D. Manning (2003). Accurate unlexicalized parsing. In *Proceedings of the*  
11532 *Association for Computational Linguistics (ACL)*, pp. 423–430.
- 11533 Klein, D. and C. D. Manning (2004). Corpus-based induction of syntactic structure: Mod-  
11534 els of dependency and constituency. In *Proceedings of the Association for Computational*  
11535 *Linguistics (ACL)*.
- 11536 Klein, G., Y. Kim, Y. Deng, J. Senellart, and A. M. Rush (2017). Opennmt: Open-source  
11537 toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*.

- 11538 Klementiev, A., I. Titov, and B. Bhattachari (2012). Inducing crosslingual distributed representations of words. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp. 1459–1474.
- 11541 Klenner, M. (2007). Enforcing consistency on coreference sets. In *Recent Advances in Natural Language Processing (RANLP)*, pp. 323–328.
- 11543 Knight, K. (1999). Decoding complexity in word-replacement translation models. *Computational Linguistics* 25(4), 607–615.
- 11545 Knight, K. and J. Graehl (1998). Machine transliteration. *Computational Linguistics* 24(4), 599–612.
- 11547 Knight, K. and D. Marcu (2000). Statistics-based summarization-step one: Sentence compression. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 703–710.
- 11550 Knight, K. and J. May (2009). Applications of weighted automata in natural language processing. In *Handbook of Weighted Automata*, pp. 571–596. Springer.
- 11552 Knott, A. (1996). *A data-driven methodology for motivating a set of coherence relations*. Ph. D. thesis, The University of Edinburgh.
- 11554 Koehn, P. (2005). Europarl: A parallel corpus for statistical machine translation. In *MT summit*, Volume 5, pp. 79–86.
- 11556 Koehn, P. (2009). *Statistical machine translation*. Cambridge University Press.
- 11557 Koehn, P. (2017). Neural machine translation. *arXiv preprint arXiv:1709.07809*.
- 11558 Konstas, I. and M. Lapata (2013). A global model for concept-to-text generation. *Journal of Artificial Intelligence Research* 48, 305–346.
- 11560 Koo, T., X. Carreras, and M. Collins (2008, jun). Simple semi-supervised dependency parsing. In *Proceedings of ACL-08: HLT*, Columbus, Ohio, pp. 595–603. Association for Computational Linguistics.
- 11563 Koo, T. and M. Collins (2005). Hidden-variable models for discriminative reranking. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 507–514.
- 11565 Koo, T. and M. Collins (2010). Efficient third-order dependency parsers. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11567 Koo, T., A. Globerson, X. Carreras, and M. Collins (2007). Structured prediction models via the matrix-tree theorem. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 141–150.

- 11570 Kovach, B. and T. Rosenstiel (2014). *The elements of journalism: What newspeople should know  
11571 and the public should expect*. Three Rivers Press.
- 11572 Krishnamurthy, J. (2016). Probabilistic models for learning a semantic parser lexicon. In  
11573 *Proceedings of the North American Chapter of the Association for Computational Linguistics  
11574 (NAACL)*, pp. 606–616.
- 11575 Krishnamurthy, J. and T. M. Mitchell (2012). Weakly supervised training of semantic  
11576 parsers. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,  
11577 pp. 754–765.
- 11578 Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep  
11579 convolutional neural networks. In *Neural Information Processing Systems (NIPS)*, pp.  
11580 1097–1105.
- 11581 Kübler, S., R. McDonald, and J. Nivre (2009). Dependency parsing. *Synthesis Lectures on  
11582 Human Language Technologies* 1(1), 1–127.
- 11583 Kuhlmann, M. and J. Nivre (2010). Transition-based techniques for non-projective depen-  
11584 dency parsing. *Northern European Journal of Language Technology (NEJLT)* 2(1), 1–19.
- 11585 Kummerfeld, J. K., T. Berg-Kirkpatrick, and D. Klein (2015). An empirical analysis of op-  
11586 timization for max-margin NLP. In *Proceedings of Empirical Methods for Natural Language  
11587 Processing (EMNLP)*.
- 11588 Kwiatkowski, T., S. Goldwater, L. Zettlemoyer, and M. Steedman (2012). A probabilistic  
11589 model of syntactic and semantic acquisition from child-directed utterances and their  
11590 meanings. In *Proceedings of the European Chapter of the Association for Computational Lin-  
11591 guistics (EACL)*, pp. 234–244.
- 11592 Lafferty, J., A. McCallum, and F. Pereira (2001). Conditional random fields: Probabilistic  
11593 models for segmenting and labeling sequence data. In *icml*.
- 11594 Lakoff, G. (1973). Hedges: A study in meaning criteria and the logic of fuzzy concepts.  
11595 *Journal of philosophical logic* 2(4), 458–508.
- 11596 Lample, G., M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer (2016). Neural  
11597 architectures for named entity recognition. In *Proceedings of the North American Chapter  
11598 of the Association for Computational Linguistics (NAACL)*, pp. 260–270.
- 11599 Langkilde, I. and K. Knight (1998). Generation that exploits corpus-based statistical  
11600 knowledge. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 704–  
11601 710.

- 11602 Lapata, M. (2003). Probabilistic text structuring: Experiments with sentence ordering. In  
 11603 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 545–552.
- 11604 Lappin, S. and H. J. Leass (1994). An algorithm for pronominal anaphora resolution.  
 11605 *Computational linguistics* 20(4), 535–561.
- 11606 Lari, K. and S. J. Young (1990). The estimation of stochastic context-free grammars using  
 11607 the inside-outside algorithm. *Computer speech & language* 4(1), 35–56.
- 11608 Lascarides, A. and N. Asher (2007). Segmented discourse representation theory: Dynamic  
 11609 semantics with discourse structure. In *Computing meaning*, pp. 87–124. Springer.
- 11610 Law, E. and L. v. Ahn (2011). Human computation. *Synthesis Lectures on Artificial Intelli-*  
 11611 *gence and Machine Learning* 5(3), 1–121.
- 11612 Lebret, R., D. Grangier, and M. Auli (2016). Neural text generation from structured data  
 11613 with application to the biography domain. In *Proceedings of Empirical Methods for Natural*  
 11614 *Language Processing (EMNLP)*, pp. 1203–1213.
- 11615 LeCun, Y. and Y. Bengio (1995). Convolutional networks for images, speech, and time  
 11616 series. *The handbook of brain theory and neural networks* 3361.
- 11617 LeCun, Y., L. Bottou, G. B. Orr, and K.-R. Müller (1998). Efficient backprop. In *Neural*  
 11618 *networks: Tricks of the trade*, pp. 9–50. Springer.
- 11619 Lee, C. M. and S. S. Narayanan (2005). Toward detecting emotions in spoken dialogs.  
 11620 *IEEE transactions on speech and audio processing* 13(2), 293–303.
- 11621 Lee, H., A. Chang, Y. Peirsman, N. Chambers, M. Surdeanu, and D. Jurafsky (2013). De-  
 11622 terministic coreference resolution based on entity-centric, precision-ranked rules. *Com-*  
 11623 *putational Linguistics* 39(4), 885–916.
- 11624 Lee, H., Y. Peirsman, A. Chang, N. Chambers, M. Surdeanu, and D. Jurafsky (2011). Stan-  
 11625 ford’s multi-pass sieve coreference resolution system at the conll-2011 shared task. In  
 11626 *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 28–34. Associa-  
 11627 tion for Computational Linguistics.
- 11628 Lee, K., L. He, M. Lewis, and L. Zettlemoyer (2017). End-to-end neural coreference reso-  
 11629 lution. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11630 Lenat, D. B., R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd (1990). Cyc: toward  
 11631 programs with common sense. *Communications of the ACM* 33(8), 30–49.
- 11632 Lesk, M. (1986). Automatic sense disambiguation using machine readable dictionaries:  
 11633 how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual interna-*  
 11634 *tional conference on Systems documentation*, pp. 24–26. ACM.

- 11635 Levesque, H. J., E. Davis, and L. Morgenstern (2011). The winograd schema challenge.  
11636 In *Aaa spring symposium: Logical formalizations of commonsense reasoning*, Volume 46, pp.  
11637 47.
- 11638 Levin, E., R. Pieraccini, and W. Eckert (1998). Using markov decision process for learning  
11639 dialogue strategies. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the*  
11640 *1998 IEEE International Conference on*, Volume 1, pp. 201–204. IEEE.
- 11641 Levy, O. and Y. Goldberg (2014). Dependency-based word embeddings. In *Proceedings of*  
11642 *the Association for Computational Linguistics (ACL)*, pp. 302–308.
- 11643 Levy, O., Y. Goldberg, and I. Dagan (2015). Improving distributional similarity with  
11644 lessons learned from word embeddings. *Transactions of the Association for Computational*  
11645 *Linguistics* 3, 211–225.
- 11646 Levy, R. and C. Manning (2009). An informal introduction to computational semantics.
- 11647 Lewis, M. and M. Steedman (2013). Combined distributional and logical semantics. *Trans-*  
11648 *actions of the Association for Computational Linguistics* 1, 179–192.
- 11649 Lewis II, P. M. and R. E. Stearns (1968). Syntax-directed transduction. *Journal of the ACM*  
11650 (*JACM*) 15(3), 465–488.
- 11651 Li, J. and D. Jurafsky (2015). Do multi-sense embeddings improve natural language  
11652 understanding? In *Proceedings of Empirical Methods for Natural Language Processing*  
11653 (*EMNLP*), pp. 1722–1732.
- 11654 Li, J. and D. Jurafsky (2017). Neural net models of open-domain discourse coherence. In *Proceed-*  
11655 *ings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 198–209.
- 11656 Li, J., R. Li, and E. Hovy (2014). Recursive deep models for discourse parsing. In *Proceed-*  
11657 *ings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11658 Li, J., M.-T. Luong, and D. Jurafsky (2015). A hierarchical neural autoencoder for para-  
11659 graphs and documents. In *Proceedings of Empirical Methods for Natural Language Process-*  
11660 *ing (EMNLP)*.
- 11661 Li, J., T. Luong, D. Jurafsky, and E. Hovy (2015). When are tree structures necessary  
11662 for deep learning of representations? In *Proceedings of Empirical Methods for Natural*  
11663 *Language Processing (EMNLP)*, pp. 2304–2314.
- 11664 Li, J., W. Monroe, A. Ritter, D. Jurafsky, M. Galley, and J. Gao (2016, November). Deep  
11665 reinforcement learning for dialogue generation. In *Proceedings of the 2016 Conference on*  
11666 *Empirical Methods in Natural Language Processing*, Austin, Texas, pp. 1192–1202. Associa-  
11667 *tion for Computational Linguistics*.

- 11668 Li, Q., S. Anzaroot, W.-P. Lin, X. Li, and H. Ji (2011). Joint inference for cross-document  
 11669 information extraction. In *Proceedings of the International Conference on Information and  
 11670 Knowledge Management (CIKM)*, pp. 2225–2228.
- 11671 Li, Q., H. Ji, and L. Huang (2013). Joint event extraction via structured prediction with  
 11672 global features. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.  
 11673 73–82.
- 11674 Liang, P., A. Bouchard-Côté, D. Klein, and B. Taskar (2006). An end-to-end discriminative  
 11675 approach to machine translation. In *Proceedings of the Association for Computational  
 11676 Linguistics (ACL)*, pp. 761–768.
- 11677 Liang, P., M. Jordan, and D. Klein (2009). Learning semantic correspondences with less  
 11678 supervision. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 91–  
 11679 99.
- 11680 Liang, P., M. I. Jordan, and D. Klein (2013). Learning dependency-based compositional  
 11681 semantics. *Computational Linguistics* 39(2), 389–446.
- 11682 Liang, P. and D. Klein (2009). Online em for unsupervised models. In *Proceedings of the  
 11683 North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 611–  
 11684 619.
- 11685 Liang, P., S. Petrov, M. I. Jordan, and D. Klein (2007). The infinite pcfg using hierarchical  
 11686 dirichlet processes. In *Proceedings of Empirical Methods for Natural Language Processing  
 11687 (EMNLP)*, pp. 688–697.
- 11688 Liang, P. and C. Potts (2015). Bringing machine learning and compositional semantics  
 11689 together. *Annual Review of Linguistics* 1(1), 355–376.
- 11690 Lieber, R. (2015). *Introducing morphology*. Cambridge University Press.
- 11691 Lin, D. (1998). Automatic retrieval and clustering of similar words. In *Proceedings of the  
 11692 17th international conference on Computational linguistics-Volume 2*, pp. 768–774. Associa-  
 11693 tion for Computational Linguistics.
- 11694 Lin, J. and C. Dyer (2010). Data-intensive text processing with mapreduce. *Synthesis  
 11695 Lectures on Human Language Technologies* 3(1), 1–177.
- 11696 Lin, Z., M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio (2017). A  
 11697 structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*.
- 11698 Lin, Z., M.-Y. Kan, and H. T. Ng (2009). Recognizing implicit discourse relations in the  
 11699 penn discourse treebank. In *Proceedings of Empirical Methods for Natural Language Pro-  
 11700 cessing (EMNLP)*, pp. 343–351.

- 11701 Lin, Z., H. T. Ng, and M.-Y. Kan (2011). Automatically evaluating text coherence using  
11702 discourse relations. In *Proceedings of the Association for Computational Linguistics (ACL)*,  
11703 pp. 997–1006.
- 11704 Lin, Z., H. T. Ng, and M. Y. Kan (2014, nov). A PDTB-styled end-to-end discourse parser.  
11705 *Natural Language Engineering FirstView*, 1–34.
- 11706 Ling, W., C. Dyer, A. Black, and I. Trancoso (2015). Two/too simple adaptations of  
11707 word2vec for syntax problems. In *Proceedings of the North American Chapter of the As-*  
11708 *sociation for Computational Linguistics (NAACL)*.
- 11709 Ling, W., T. Luís, L. Marujo, R. F. Astudillo, S. Amir, C. Dyer, A. W. Black, and I. Trancoso  
11710 (2015). Finding function in form: Compositional character models for open vocabulary  
11711 word representation. In *Proceedings of Empirical Methods for Natural Language Processing*  
11712 (*EMNLP*).
- 11713 Ling, W., G. Xiang, C. Dyer, A. Black, and I. Trancoso (2013). Microblogs as parallel cor-  
11714 pora. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11715 Ling, X., S. Singh, and D. S. Weld (2015). Design challenges for entity linking. *Transactions*  
11716 *of the Association for Computational Linguistics* 3, 315–328.
- 11717 Linguistic Data Consortium (2005, July). ACE (automatic content extraction) English an-  
11718 notation guidelines for relations. Technical Report Version 5.8.3, Linguistic Data Con-  
11719 sortium.
- 11720 Liu, B. (2015). *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. Cambridge  
11721 University Press.
- 11722 Liu, D. C. and J. Nocedal (1989). On the limited memory BFGS method for large scale  
11723 optimization. *Mathematical programming* 45(1-3), 503–528.
- 11724 Liu, Y., Q. Liu, and S. Lin (2006). Tree-to-string alignment template for statistical machine  
11725 translation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 609–  
11726 616.
- 11727 Loper, E. and S. Bird (2002). Nltk: The natural language toolkit. In *Proceedings of the ACL-*  
11728 *02 Workshop on Effective tools and methodologies for teaching natural language processing and*  
11729 *computational linguistics-Volume 1*, pp. 63–70. Association for Computational Linguistics.
- 11730 Louis, A., A. Joshi, and A. Nenkova (2010). Discourse indicators for content selection in  
11731 summarization. In *Proceedings of the 11th Annual Meeting of the Special Interest Group on*  
11732 *Discourse and Dialogue*, pp. 147–156. Association for Computational Linguistics.

- 11733 Louis, A. and A. Nenkova (2013). What makes writing great? first experiments on article  
 11734 quality prediction in the science journalism domain. *Transactions of the Association for  
 11735 Computational Linguistics* 1, 341–352.
- 11736 Loveland, D. W. (2016). *Automated Theorem Proving: a logical basis*. Elsevier.
- 11737 Lowe, R., N. Pow, I. V. Serban, and J. Pineau (2015). The ubuntu dialogue corpus: A large  
 11738 dataset for research in unstructured multi-turn dialogue systems. In *Proceedings of the  
 11739 Special Interest Group on Discourse and Dialogue (SIGDIAL)*.
- 11740 Luo, X. (2005). On coreference resolution performance metrics. In *Proceedings of Empirical  
 11741 Methods for Natural Language Processing (EMNLP)*, pp. 25–32.
- 11742 Luo, X., A. Ittycheriah, H. Jing, N. Kambhatla, and S. Roukos (2004). A mention-  
 11743 synchronous coreference resolution algorithm based on the bell tree. In *Proceedings  
 11744 of the Association for Computational Linguistics (ACL)*.
- 11745 Luong, M.-T., R. Socher, and C. D. Manning (2013). Better word representations with  
 11746 recursive neural networks for morphology. *CoNLL-2013* 104.
- 11747 Luong, T., H. Pham, and C. D. Manning (2015). Effective approaches to attention-based  
 11748 neural machine translation. In *Proceedings of Empirical Methods for Natural Language  
 11749 Processing (EMNLP)*, pp. 1412–1421.
- 11750 Luong, T., I. Sutskever, Q. Le, O. Vinyals, and W. Zaremba (2015). Addressing the rare  
 11751 word problem in neural machine translation. In *Proceedings of the Association for Compu-  
 11752 tational Linguistics (ACL)*, pp. 11–19.
- 11753 Maas, A. L., A. Y. Hannun, and A. Y. Ng (2013). Rectifier nonlinearities improve neu-  
 11754 ral network acoustic models. In *Proceedings of the International Conference on Machine  
 11755 Learning (ICML)*.
- 11756 Mairesse, F. and M. A. Walker (2011). Controlling user perceptions of linguistic style:  
 11757 Trainable generation of personality traits. *Computational Linguistics* 37(3), 455–488.
- 11758 Mani, I., M. Verhagen, B. Wellner, C. M. Lee, and J. Pustejovsky (2006). Machine learning  
 11759 of temporal relations. In *Proceedings of the Association for Computational Linguistics (ACL)*,  
 11760 pp. 753–760.
- 11761 Mann, W. C. and S. A. Thompson (1988). Rhetorical structure theory: Toward a functional  
 11762 theory of text organization. *Text* 8(3), 243–281.
- 11763 Manning, C. D. (2015). Computational linguistics and deep learning. *Computational Lin-  
 11764 guistics* 41(4), 701–707.

- 11765 Manning, C. D. (2016). Computational linguistics and deep learning. *Computational Lin-*  
11766 *guistics* 41(4).
- 11767 Manning, C. D., P. Raghavan, H. Schütze, et al. (2008). *Introduction to information retrieval*,  
11768 Volume 1. Cambridge university press.
- 11769 Manning, C. D. and H. Schütze (1999). *Foundations of Statistical Natural Language Process-*  
11770 *ing*. Cambridge, Massachusetts: MIT press.
- 11771 Marcu, D. (1996). Building up rhetorical structure trees. In *Proceedings of the National*  
11772 *Conference on Artificial Intelligence*, pp. 1069–1074.
- 11773 Marcu, D. (1997a). From discourse structures to text summaries. In *Proceedings of the*  
11774 *workshop on Intelligent Scalable Text Summarization*.
- 11775 Marcu, D. (1997b). From local to global coherence: A bottom-up approach to text plan-  
11776 ning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 629–635.
- 11777 Marcus, M. P., M. A. Marcinkiewicz, and B. Santorini (1993). Building a large annotated  
11778 corpus of English: The Penn Treebank. *Computational Linguistics* 19(2), 313–330.
- 11779 Maron, O. and T. Lozano-Pérez (1998). A framework for multiple-instance learning. In  
11780 *Neural Information Processing Systems (NIPS)*, pp. 570–576.
- 11781 Márquez, G. G. (1970). *One Hundred Years of Solitude*. Harper & Row. English translation  
11782 by Gregory Rabassa.
- 11783 Martins, A. F. T., N. A. Smith, and E. P. Xing (2009). Concise integer linear programming  
11784 formulations for dependency parsing. In *Proceedings of the Association for Computational*  
11785 *Linguistics (ACL)*, pp. 342–350.
- 11786 Martins, A. F. T., N. A. Smith, E. P. Xing, P. M. Q. Aguiar, and M. A. T. Figueiredo (2010).  
11787 Turbo parsers: Dependency parsing by approximate variational inference. In *Proceed-  
11788 ings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 34–44.
- 11789 Matsuzaki, T., Y. Miyao, and J. Tsujii (2005). Probabilistic cfg with latent annotations. In  
11790 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 75–82.
- 11791 Matthiessen, C. and J. A. Bateman (1991). *Text generation and systemic-functional linguistics:  
11792 experiences from English and Japanese*. Pinter Publishers.
- 11793 McCallum, A. and W. Li (2003). Early results for named entity recognition with condi-  
11794 tional random fields, feature induction and web-enhanced lexicons. In *Proceedings of  
11795 the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp.  
11796 188–191.

- 11797 McCallum, A. and B. Wellner (2004). Conditional models of identity uncertainty with  
 11798 application to noun coreference. In *NIPS*, pp. 905–912.
- 11799 McDonald, R., K. Crammer, and F. Pereira (2005). Online large-margin training of depen-  
 11800 dency parsers. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.  
 11801 91–98.
- 11802 McDonald, R., K. Hannan, T. Neylon, M. Wells, and J. Reynar (2007). Structured models  
 11803 for fine-to-coarse sentiment analysis. In *Proceedings of ACL*.
- 11804 McDonald, R. and F. Pereira (2006). Online learning of approximate dependency parsing  
 11805 algorithms. In *Proceedings of the European Chapter of the Association for Computational  
 11806 Linguistics (EACL)*.
- 11807 McKeown, K. (1992). *Text generation*. Cambridge University Press.
- 11808 McKeown, K., S. Rosenthal, K. Thadani, and C. Moore (2010). Time-efficient creation of  
 11809 an accurate sentence fusion corpus. In *Proceedings of the North American Chapter of the  
 11810 Association for Computational Linguistics (NAACL)*, pp. 317–320.
- 11811 McKeown, K. R., R. Barzilay, D. Evans, V. Hatzivassiloglou, J. L. Klavans, A. Nenkova,  
 11812 C. Sable, B. Schiffman, and S. Sigelman (2002). Tracking and summarizing news on a  
 11813 daily basis with columbia’s newsblaster. In *Proceedings of the second international confer-  
 11814 ence on Human Language Technology Research*, pp. 280–285.
- 11815 McNamee, P. and H. T. Dang (2009). Overview of the tac 2009 knowledge base population  
 11816 track. In *Text Analysis Conference (TAC)*, Volume 17, pp. 111–113.
- 11817 Medlock, B. and T. Briscoe (2007). Weakly supervised learning for hedge classification in  
 11818 scientific literature. In *Proceedings of the Association for Computational Linguistics (ACL)*,  
 11819 pp. 992–999.
- 11820 Mei, H., M. Bansal, and M. R. Walter (2016). What to talk about and how? selective gen-  
 11821 eration using lstms with coarse-to-fine alignment. In *Proceedings of the North American  
 11822 Chapter of the Association for Computational Linguistics (NAACL)*, pp. 720–730.
- 11823 Merity, S., N. S. Keskar, and R. Socher (2018). Regularizing and optimizing lstm language  
 11824 models. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 11825 Merity, S., C. Xiong, J. Bradbury, and R. Socher (2017). Pointer sentinel mixture models.  
 11826 In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 11827 Messud, C. (2014, June). A new ‘l’étranger’. *New York Review of Books*.

- 11828 Miao, Y. and P. Blunsom (2016). Language as a latent variable: Discrete generative mod-  
11829     els for sentence compression. In *Proceedings of Empirical Methods for Natural Language*  
11830     *Processing (EMNLP)*, pp. 319–328.
- 11831 Miao, Y., L. Yu, and P. Blunsom (2016). Neural variational inference for text processing. In  
11832     *Proceedings of the International Conference on Machine Learning (ICML)*.
- 11833 Mihalcea, R., T. A. Chklovski, and A. Kilgarriff (2004, July). The senseval-3 english lexical  
11834     sample task. In *Proceedings of SENSEVAL-3*, Barcelona, Spain, pp. 25–28. Association for  
11835     Computational Linguistics.
- 11836 Mihalcea, R. and D. Radev (2011). *Graph-based natural language processing and information*  
11837     *retrieval*. Cambridge University Press.
- 11838 Mikolov, T., K. Chen, G. Corrado, and J. Dean (2013). Efficient estimation of word repre-  
11839     sentations in vector space. In *Proceedings of International Conference on Learning Represen-*  
11840     *tations*.
- 11841 Mikolov, T., A. Deoras, D. Povey, L. Burget, and J. Cernocky (2011). Strategies for train-  
11842     ing large scale neural network language models. In *Automatic Speech Recognition and*  
11843     *Understanding (ASRU), 2011 IEEE Workshop on*, pp. 196–201. IEEE.
- 11844 Mikolov, T., M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur (2010). Recurrent  
11845     neural network based language model. In *INTERSPEECH*, pp. 1045–1048.
- 11846 Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013). Distributed rep-  
11847     resentations of words and phrases and their compositionality. In *Advances in Neural*  
11848     *Information Processing Systems*, pp. 3111–3119.
- 11849 Mikolov, T., W.-t. Yih, and G. Zweig (2013). Linguistic regularities in continuous space  
11850     word representations. In *Proceedings of the North American Chapter of the Association for*  
11851     *Computational Linguistics (NAACL)*, pp. 746–751.
- 11852 Mikolov, T. and G. Zweig. Context dependent recurrent neural network language model.  
11853     In *Proceedings of Spoken Language Technology (SLT)*, pp. 234–239.
- 11854 Miller, G. A., G. A. Heise, and W. Lichten (1951). The intelligibility of speech as a function  
11855     of the context of the test materials. *Journal of experimental psychology* 41(5), 329.
- 11856 Miller, M., C. Sathi, D. Wiesenthal, J. Leskovec, and C. Potts (2011). Sentiment flow  
11857     through hyperlink networks. In *Proceedings of the International Conference on Web and*  
11858     *Social Media (ICWSM)*.
- 11859 Miller, S., J. Guinness, and A. Zamanian (2004). Name tagging with word clusters and  
11860     discriminative training. In *Proceedings of the North American Chapter of the Association for*  
11861     *Computational Linguistics (NAACL)*, pp. 337–342.

- 11862 Milne, D. and I. H. Witten (2008). Learning to link with wikipedia. In *Proceedings of the  
11863 International Conference on Information and Knowledge Management (CIKM)*, pp. 509–518.  
11864 ACM.
- 11865 Miltzakaki, E. and K. Kukich (2004). Evaluation of text coherence for electronic essay  
11866 scoring systems. *Natural Language Engineering* 10(1), 25–55.
- 11867 Minka, T. P. (1999). From hidden markov models to linear dynamical systems. Tech. Rep.  
11868 531, Vision and Modeling Group of Media Lab, MIT.
- 11869 Minsky, M. (1974). A framework for representing knowledge. Technical Report 306, MIT  
11870 AI Laboratory.
- 11871 Minsky, M. and S. Papert (1969). *Perceptrons*. MIT press.
- 11872 Mintz, M., S. Bills, R. Snow, and D. Jurafsky (2009). Distant supervision for relation extrac-  
11873 tion without labeled data. In *Proceedings of the Association for Computational Linguistics  
11874 (ACL)*, pp. 1003–1011.
- 11875 Mirza, P., R. Sprugnoli, S. Tonelli, and M. Speranza (2014). Annotating causality in the  
11876 tempeval-3 corpus. In *Proceedings of the EACL 2014 Workshop on Computational Ap-  
11877 proaches to Causality in Language (CAtoCL)*, pp. 10–19.
- 11878 Misra, D. K. and Y. Artzi (2016). Neural shift-reduce ccg semantic parsing. In *Proceedings  
11879 of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11880 Mitchell, J. and M. Lapata (2010). Composition in distributional models of semantics.  
11881 *Cognitive Science* 34(8), 1388–1429.
- 11882 Miwa, M. and M. Bansal (2016). End-to-end relation extraction using lstms on sequences  
11883 and tree structures. In *Proceedings of the Association for Computational Linguistics (ACL)*,  
11884 pp. 1105–1116.
- 11885 Mnih, A. and G. Hinton (2007). Three new graphical models for statistical language mod-  
11886 ellling. In *Proceedings of the 24th international conference on Machine learning*, ICML '07,  
11887 New York, NY, USA, pp. 641–648. ACM.
- 11888 Mnih, A. and G. E. Hinton (2008). A scalable hierarchical distributed language model. In  
11889 *Neural Information Processing Systems (NIPS)*, pp. 1081–1088.
- 11890 Mnih, A. and Y. W. Teh (2012). A fast and simple algorithm for training neural probabilis-  
11891 tic language models. In *Proceedings of the International Conference on Machine Learning  
11892 (ICML)*.
- 11893 Mohammad, S. M. and P. D. Turney (2013). Crowdsourcing a word–emotion association  
11894 lexicon. *Computational Intelligence* 29(3), 436–465.

- 11895 Mohri, M., F. Pereira, and M. Riley (2002). Weighted finite-state transducers in speech  
11896 recognition. *Computer Speech & Language* 16(1), 69–88.
- 11897 Mohri, M., A. Rostamizadeh, and A. Talwalkar (2012). *Foundations of machine learning*.  
11898 MIT press.
- 11899 Montague, R. (1973). The proper treatment of quantification in ordinary english. In *Ap-*  
11900 *proaches to natural language*, pp. 221–242. Springer.
- 11901 Moore, J. D. and C. L. Paris (1993, dec). Planning text for advisory dialogues: Capturing  
11902 intentional and rhetorical information. *Comput. Linguist.* 19(4), 651–694.
- 11903 Morante, R. and E. Blanco (2012). \*sem 2012 shared task: Resolving the scope and fo-  
11904 cus of negation. In *Proceedings of the First Joint Conference on Lexical and Computational*  
11905 *Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2:*  
11906 *Proceedings of the Sixth International Workshop on Semantic Evaluation*, pp. 265–274. Asso-  
11907 ciation for Computational Linguistics.
- 11908 Morante, R. and W. Daelemans (2009). Learning the scope of hedge cues in biomedical  
11909 texts. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language*  
11910 *Processing*, pp. 28–36. Association for Computational Linguistics.
- 11911 Morante, R. and C. Sporleder (2012). Modality and negation: An introduction to the  
11912 special issue. *Computational linguistics* 38(2), 223–260.
- 11913 Mostafazadeh, N., A. Grelish, N. Chambers, J. Allen, and L. Vanderwende (2016, June).  
11914 Caters: Causal and temporal relation scheme for semantic annotation of event struc-  
11915 tures. In *Proceedings of the Fourth Workshop on Events*, San Diego, California, pp. 51–61.  
11916 Association for Computational Linguistics.
- 11917 Mueller, T., H. Schmid, and H. Schütze (2013). Efficient higher-order CRFs for morpholog-  
11918 ical tagging. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,  
11919 pp. 322–332.
- 11920 Müller, C. and M. Strube (2006). Multi-level annotation of linguistic data with mmax2.  
11921 *Corpus technology and language pedagogy: New resources, new tools, new methods* 3, 197–  
11922 214.
- 11923 Muralidharan, A. and M. A. Hearst (2013). Supporting exploratory text analysis in litera-  
11924 ture study. *Literary and linguistic computing* 28(2), 283–295.
- 11925 Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.
- 11926 Nakagawa, T., K. Inui, and S. Kurohashi (2010). Dependency tree-based sentiment classi-  
11927 fication using crfs with hidden variables. In *Proceedings of the North American Chapter of*  
11928 *the Association for Computational Linguistics (NAACL)*, pp. 786–794.

- 11929 Nakazawa, T., M. Yaguchi, K. Uchimoto, M. Utiyama, E. Sumita, S. Kurohashi, and H. Isahara (2016). ASPEC: Asian scientific paper excerpt corpus. In *Proceedings of the Language Resources and Evaluation Conference*, pp. 2204–2208.
- 11932 Navigli, R. (2009). Word sense disambiguation: A survey. *ACM Computing Surveys (CSUR)* 41(2), 10.
- 11934 Neal, R. M. and G. E. Hinton (1998). A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pp. 355–368. Springer.
- 11936 Nenkova, A. and K. McKeown (2012). A survey of text summarization techniques. In *Mining text data*, pp. 43–76. Springer.
- 11938 Neubig, G. (2017). Neural machine translation and sequence-to-sequence models: A tutorial. *arXiv preprint arXiv:1703.01619*.
- 11940 Neubig, G., C. Dyer, Y. Goldberg, A. Matthews, W. Ammar, A. Anastasopoulos, M. Balles-teros, D. Chiang, D. Clothiaux, T. Cohn, K. Duh, M. Faruqui, C. Gan, D. Garrette, Y. Ji, L. Kong, A. Kuncoro, G. Kumar, C. Malaviya, P. Michel, Y. Oda, M. Richardson, N. Saphra, S. Swayamdipta, and P. Yin (2017). Dynet: The dynamic neural network toolkit.
- 11945 Neubig, G., Y. Goldberg, and C. Dyer (2017). On-the-fly operation batching in dynamic computation graphs. In *Neural Information Processing Systems (NIPS)*.
- 11947 Neuhaus, P. and N. Bröker (1997). The complexity of recognition of linguistically adequate dependency grammars. In *eacl*, pp. 337–343.
- 11949 Newman, D., C. Chemudugunta, and P. Smyth (2006). Statistical entity-topic models. In *Proceedings of Knowledge Discovery and Data Mining (KDD)*, pp. 680–686.
- 11951 Ng, V. (2010). Supervised noun phrase coreference research: The first fifteen years. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pp. 1396–1411. Association for Computational Linguistics.
- 11954 Nguyen, D. and A. S. Dogruöz (2013). Word level language identification in online multi-lingual communication. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11957 Nguyen, D. T. and S. Joty (2017). A neural local coherence model. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1320–1330.
- 11959 Nigam, K., A. K. McCallum, S. Thrun, and T. Mitchell (2000). Text classification from labeled and unlabeled documents using em. *Machine learning* 39(2-3), 103–134.

- 11961 Nirenburg, S. and Y. Wilks (2001). What's in a symbol: ontology, representation and lan-  
11962 guage. *Journal of Experimental & Theoretical Artificial Intelligence* 13(1), 9–23.
- 11963 Nivre, J. (2008). Algorithms for deterministic incremental dependency parsing. *Computa-*  
11964 *tional Linguistics* 34(4), 513–553.
- 11965 Nivre, J., M.-C. de Marneffe, F. Ginter, Y. Goldberg, J. Hajič, C. D. Manning, R. McDonald,  
11966 S. Petrov, S. Pyysalo, N. Silveira, R. Tsarfaty, and D. Zeman (2016, may). Universal de-  
11967 pendencies v1: A multilingual treebank collection. In N. C. C. Chair), K. Choukri, T. De-  
11968 clerck, S. Goggi, M. Grobelnik, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk,  
11969 and S. Piperidis (Eds.), *Proceedings of the Tenth International Conference on Language Re-*  
11970 *sources and Evaluation (LREC 2016)*, Paris, France. European Language Resources Asso-  
11971 ciation (ELRA).
- 11972 Nivre, J. and J. Nilsson (2005). Pseudo-projective dependency parsing. In *Proceedings of the*  
11973 *43rd Annual Meeting on Association for Computational Linguistics*, pp. 99–106. Association  
11974 for Computational Linguistics.
- 11975 Novikoff, A. B. (1962). On convergence proofs on perceptrons. In *Proceedings of the Sym-*  
11976 *posium on the Mathematical Theory of Automata*, Volume 12, pp. 615—622.
- 11977 Och, F. J. and H. Ney (2003). A systematic comparison of various statistical alignment  
11978 models. *Computational linguistics* 29(1), 19–51.
- 11979 O'Connor, B., M. Krieger, and D. Ahn (2010). Tweetmotif: Exploratory search and topic  
11980 summarization for twitter. In *Proceedings of the International Conference on Web and Social*  
11981 *Media (ICWSM)*, pp. 384–385.
- 11982 Oflazer, K. and İ. Kuruöz (1994). Tagging and morphological disambiguation of turkish  
11983 text. In *Proceedings of the fourth conference on Applied natural language processing*, pp. 144–  
11984 149. Association for Computational Linguistics.
- 11985 Ohta, T., Y. Tateisi, and J.-D. Kim (2002). The genia corpus: An annotated research abstract  
11986 corpus in molecular biology domain. In *Proceedings of the second international conference*  
11987 *on Human Language Technology Research*, pp. 82–86. Morgan Kaufmann Publishers Inc.
- 11988 Onishi, T., H. Wang, M. Bansal, K. Gimpel, and D. McAllester (2016). Who did what: A  
11989 large-scale person-centered cloze dataset. In *Proceedings of Empirical Methods for Natural*  
11990 *Language Processing (EMNLP)*, pp. 2230–2235.
- 11991 Owoputi, O., B. O'Connor, C. Dyer, K. Gimpel, N. Schneider, and N. A. Smith (2013).  
11992 Improved part-of-speech tagging for online conversational text with word clusters. In  
11993 *Proceedings of the North American Chapter of the Association for Computational Linguistics*  
11994 (*NAACL*), pp. 380–390.

- 11995 Packard, W., E. M. Bender, J. Read, S. Oepen, and R. Dridan (2014). Simple negation  
11996 scope resolution through deep parsing: A semantic solution to a semantic problem. In  
11997 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 69–78.
- 11998 Paice, C. D. (1990). Another stemmer. In *ACM SIGIR Forum*, Volume 24, pp. 56–61.
- 11999 Pak, A. and P. Paroubek (2010). Twitter as a corpus for sentiment analysis and opinion  
12000 mining. In *LREC*, Volume 10, pp. 1320–1326.
- 12001 Palmer, F. R. (2001). *Mood and modality*. Cambridge University Press.
- 12002 Palmer, M., D. Gildea, and P. Kingsbury (2005). The proposition bank: An annotated  
12003 corpus of semantic roles. *Computational linguistics* 31(1), 71–106.
- 12004 Pan, S. J. and Q. Yang (2010). A survey on transfer learning. *IEEE Transactions on knowledge  
12005 and data engineering* 22(10), 1345–1359.
- 12006 Pan, X., T. Cassidy, U. Hermjakob, H. Ji, and K. Knight (2015). Unsupervised entity linking  
12007 with abstract meaning representation. In *Proceedings of the North American Chapter of the  
12008 Association for Computational Linguistics (NAACL)*, pp. 1130–1139.
- 12009 Pang, B. and L. Lee (2004). A sentimental education: Sentiment analysis using subjectivity  
12010 summarization based on minimum cuts. In *Proceedings of the Association for Compu-  
12011 tational Linguistics (ACL)*, pp. 271–278.
- 12012 Pang, B. and L. Lee (2005). Seeing stars: Exploiting class relationships for sentiment cate-  
12013 gorization with respect to rating scales. In *Proceedings of the Association for Computational  
12014 Linguistics (ACL)*, pp. 115–124.
- 12015 Pang, B. and L. Lee (2008). Opinion mining and sentiment analysis. *Foundations and trends  
12016 in information retrieval* 2(1–2), 1–135.
- 12017 Pang, B., L. Lee, and S. Vaithyanathan (2002). Thumbs up?: sentiment classification using  
12018 machine learning techniques. In *Proceedings of Empirical Methods for Natural Language  
12019 Processing (EMNLP)*, pp. 79–86.
- 12020 Papineni, K., S. Roukos, T. Ward, and W.-J. Zhu (2002). Bleu: a method for automatic  
12021 evaluation of machine translation. In *Proceedings of the Association for Computational  
12022 Linguistics (ACL)*, pp. 311–318.
- 12023 Park, J. and C. Cardie (2012). Improving implicit discourse relation recognition through  
12024 feature set optimization. In *Proceedings of the Special Interest Group on Discourse and Dia-  
12025 logue (SIGDIAL)*, pp. 108–112.
- 12026 Parsons, T. (1990). *Events in the Semantics of English*, Volume 5. MIT Press.

- 12027 Pascanu, R., T. Mikolov, and Y. Bengio (2013). On the difficulty of training recurrent neural  
12028 networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp.  
12029 1310–1318.
- 12030 Paul, M., M. Federico, and S. Stüker (2010). Overview of the iwslt 2010 evaluation cam-  
12031 paign. In *International Workshop on Spoken Language Translation (IWSLT) 2010*.
- 12032 Pedersen, T., S. Patwardhan, and J. Michelizzi (2004). Wordnet::similarity - measuring the  
12033 relatedness of concepts. In *Proceedings of the North American Chapter of the Association for*  
12034 *Computational Linguistics (NAACL)*, pp. 38–41.
- 12035 Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blon-  
12036 del, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau,  
12037 M. Brucher, M. Perrot, and E. Duchesnay (2011). Scikit-learn: Machine learning in  
12038 Python. *Journal of Machine Learning Research* 12, 2825–2830.
- 12039 Pei, W., T. Ge, and B. Chang (2015). An effective neural network model for graph-based  
12040 dependency parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*,  
12041 pp. 313–322.
- 12042 Peldszus, A. and M. Stede (2013). From argument diagrams to argumentation mining  
12043 in texts: A survey. *International Journal of Cognitive Informatics and Natural Intelligence*  
12044 (*IJCINI*) 7(1), 1–31.
- 12045 Peldszus, A. and M. Stede (2015). An annotated corpus of argumentative microtexts. In  
12046 *Proceedings of the First Conference on Argumentation*.
- 12047 Peng, F., F. Feng, and A. McCallum (2004). Chinese segmentation and new word detec-  
12048 tion using conditional random fields. In *Proceedings of the International Conference on*  
12049 *Computational Linguistics (COLING)*, pp. 562.
- 12050 Pennington, J., R. Socher, and C. Manning (2014). Glove: Global vectors for word repre-  
12051 sentation. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,  
12052 pp. 1532–1543.
- 12053 Pereira, F. and Y. Schabes (1992). Inside-outside reestimation from partially bracketed  
12054 corpora. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 128–  
12055 135.
- 12056 Pereira, F. C. N. and S. M. Shieber (2002). *Prolog and natural-language analysis*. Microtome  
12057 Publishing.
- 12058 Peters, M. E., M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer  
12059 (2018). Deep contextualized word representations. In *Proceedings of the North American*  
12060 *Chapter of the Association for Computational Linguistics (NAACL)*.

- 12061 Peterson, W. W., T. G. Birdsall, and W. C. Fox (1954). The theory of signal detectability.  
12062     *Transactions of the IRE professional group on information theory* 4(4), 171–212.
- 12063 Petrov, S., L. Barrett, R. Thibaux, and D. Klein (2006). Learning accurate, compact, and in-  
12064     terpretable tree annotation. In *Proceedings of the Association for Computational Linguistics*  
12065     (*ACL*).
- 12066 Petrov, S., D. Das, and R. McDonald (2012, May). A universal part-of-speech tagset. In  
12067     *Proceedings of LREC*.
- 12068 Petrov, S. and R. McDonald (2012). Overview of the 2012 shared task on parsing the web.  
12069     In *Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL)*,  
12070     Volume 59.
- 12071 Pinker, S. (2003). *The language instinct: How the mind creates language*. Penguin UK.
- 12072 Pinter, Y., R. Guthrie, and J. Eisenstein (2017). Mimicking word embeddings using  
12073     subword RNNs. In *Proceedings of Empirical Methods for Natural Language Processing*  
12074     (*EMNLP*).
- 12075 Pitler, E., A. Louis, and A. Nenkova (2009). Automatic sense prediction for implicit dis-  
12076     course relations in text. In *Proceedings of the Association for Computational Linguistics*  
12077     (*ACL*).
- 12078 Pitler, E. and A. Nenkova (2009). Using syntax to disambiguate explicit discourse con-  
12079     nectives in text. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.  
12080     13–16.
- 12081 Pitler, E., M. Raghupathy, H. Mehta, A. Nenkova, A. Lee, and A. Joshi (2008). Easily iden-  
12082     tifiable discourse relations. In *Proceedings of the International Conference on Computational*  
12083     *Linguistics (COLING)*, pp. 87–90.
- 12084 Plank, B., A. Søgaard, and Y. Goldberg (2016). Multilingual part-of-speech tagging with  
12085     bidirectional long short-term memory models and auxiliary loss. In *Proceedings of the*  
12086     *Association for Computational Linguistics (ACL)*.
- 12087 Poesio, M., R. Stevenson, B. Di Eugenio, and J. Hitzeman (2004). Centering: A parametric  
12088     theory and its instantiations. *Computational linguistics* 30(3), 309–363.
- 12089 Polanyi, L. and A. Zaenen (2006). Contextual valence shifters. In *Computing attitude and*  
12090     *affect in text: Theory and applications*. Springer.
- 12091 Ponzetto, S. P. and M. Strube (2006). Exploiting semantic role labeling, wordnet and  
12092     wikipedia for coreference resolution. In *Proceedings of the North American Chapter of*  
12093     *the Association for Computational Linguistics (NAACL)*, pp. 192–199.

- 12094 Ponzetto, S. P. and M. Strube (2007). Knowledge derived from wikipedia for computing  
12095 semantic relatedness. *Journal of Artificial Intelligence Research* 30, 181–212.
- 12096 Poon, H. and P. Domingos (2008). Joint unsupervised coreference resolution with markov  
12097 logic. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.  
12098 650–659.
- 12099 Poon, H. and P. Domingos (2009). Unsupervised semantic parsing. In *Proceedings of Em-  
12100 pirical Methods for Natural Language Processing (EMNLP)*, pp. 1–10.
- 12101 Popel, M., D. Marecek, J. Stepánek, D. Zeman, and Z. Zabokrtský (2013). Coordination  
12102 structures in dependency treebanks. In *Proceedings of the Association for Computational  
12103 Linguistics (ACL)*, pp. 517–527.
- 12104 Popescu, A.-M., O. Etzioni, and H. Kautz (2003). Towards a theory of natural language  
12105 interfaces to databases. In *Proceedings of Intelligent User Interfaces (IUI)*, pp. 149–157.
- 12106 Poplack, S. (1980). Sometimes i'll start a sentence in spanish y termino en español: toward  
12107 a typology of code-switching1. *Linguistics* 18(7-8), 581–618.
- 12108 Porter, M. F. (1980). An algorithm for suffix stripping. *Program* 14(3), 130–137.
- 12109 Prabhakaran, V., O. Rambow, and M. Diab (2010). Automatic committed belief tagging.  
12110 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.  
12111 1014–1022.
- 12112 Pradhan, S., X. Luo, M. Recasens, E. Hovy, V. Ng, and M. Strube (2014). Scoring corefer-  
12113 ence partitions of predicted mentions: A reference implementation. In *Proceedings of the  
12114 Association for Computational Linguistics (ACL)*, pp. 30–35.
- 12115 Pradhan, S., L. Ramshaw, M. Marcus, M. Palmer, R. Weischedel, and N. Xue (2011).  
12116 CoNLL-2011 shared task: Modeling unrestricted coreference in OntoNotes. In *Proceed-  
12117 ings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*,  
12118 pp. 1–27. Association for Computational Linguistics.
- 12119 Pradhan, S., W. Ward, K. Hacioglu, J. H. Martin, and D. Jurafsky (2005). Semantic role  
12120 labeling using different syntactic views. In *Proceedings of the Association for Computational  
12121 Linguistics (ACL)*, pp. 581–588.
- 12122 Prasad, R., N. Dinesh, A. Lee, E. Miltsakaki, L. Robaldo, A. Joshi, and B. Webber (2008).  
12123 The Penn Discourse Treebank 2.0. In *Proceedings of LREC*.
- 12124 Punyakanok, V., D. Roth, and W.-t. Yih (2008). The importance of syntactic parsing and  
12125 inference in semantic role labeling. *Computational Linguistics* 34(2), 257–287.

- 12126 Pustejovsky, J., P. Hanks, R. Sauri, A. See, R. Gaizauskas, A. Setzer, D. Radev, B. Sundheim,  
 12127 D. Day, L. Ferro, et al. (2003). The timebank corpus. In *Corpus linguistics*, Volume 2003,  
 12128 pp. 40. Lancaster, UK.
- 12129 Pustejovsky, J., B. Ingria, R. Sauri, J. Castano, J. Littman, R. Gaizauskas, A. Setzer, G. Katz,  
 12130 and I. Mani (2005). The specification language timeml. In *The language of time: A reader*,  
 12131 pp. 545–557. Oxford University Press.
- 12132 Qin, L., Z. Zhang, H. Zhao, Z. Hu, and E. Xing (2017). Adversarial connective-exploiting  
 12133 networks for implicit discourse relation classification. In *Proceedings of the Association  
 12134 for Computational Linguistics (ACL)*, pp. 1006–1017.
- 12135 Qiu, G., B. Liu, J. Bu, and C. Chen (2011). Opinion word expansion and target extraction  
 12136 through double propagation. *Computational linguistics* 37(1), 9–27.
- 12137 Quattoni, A., S. Wang, L.-P. Morency, M. Collins, and T. Darrell (2007). Hidden conditional  
 12138 random fields. *IEEE transactions on pattern analysis and machine intelligence* 29(10).
- 12139 Rahman, A. and V. Ng (2011). Narrowing the modeling gap: a cluster-ranking approach  
 12140 to coreference resolution. *Journal of Artificial Intelligence Research* 40, 469–521.
- 12141 Rajpurkar, P., J. Zhang, K. Lopyrev, and P. Liang (2016). Squad: 100,000+ questions for  
 12142 machine comprehension of text. In *Proceedings of Empirical Methods for Natural Language  
 12143 Processing (EMNLP)*, pp. 2383–2392.
- 12144 Ranzato, M., S. Chopra, M. Auli, and W. Zaremba (2016). Sequence level training with  
 12145 recurrent neural networks. In *Proceedings of the International Conference on Learning Rep-  
 12146 resentations (ICLR)*.
- 12147 Rao, D., D. Yarowsky, A. Shreevats, and M. Gupta (2010). Classifying latent user attributes  
 12148 in twitter. In *Proceedings of Workshop on Search and mining user-generated contents*.
- 12149 Ratinov, L. and D. Roth (2009). Design challenges and misconceptions in named entity  
 12150 recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language  
 12151 Learning*, pp. 147–155. Association for Computational Linguistics.
- 12152 Ratinov, L., D. Roth, D. Downey, and M. Anderson (2011). Local and global algorithms  
 12153 for disambiguation to wikipedia. In *Proceedings of the Association for Computational Lin-  
 12154 guistics (ACL)*, pp. 1375–1384.
- 12155 Ratliff, N. D., J. A. Bagnell, and M. Zinkevich (2007). (approximate) subgradient methods  
 12156 for structured prediction. In *Proceedings of Artificial Intelligence and Statistics (AISTATS)*,  
 12157 pp. 380–387.

- 12158 Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In *emnlp*,  
12159 pp. 133–142.
- 12160 Ratnaparkhi, A., J. Reynar, and S. Roukos (1994). A maximum entropy model for preposi-  
12161 tional phrase attachment. In *Proceedings of the workshop on Human Language Technology*,  
12162 pp. 250–255. Association for Computational Linguistics.
- 12163 Read, J. (2005). Using emoticons to reduce dependency in machine learning techniques for  
12164 sentiment classification. In *Proceedings of the ACL student research workshop*, pp. 43–48.  
12165 Association for Computational Linguistics.
- 12166 Reisinger, D., R. Rudinger, F. Ferraro, C. Harman, K. Rawlins, and B. V. Durme (2015).  
12167 Semantic proto-roles. *Transactions of the Association for Computational Linguistics* 3, 475–  
12168 488.
- 12169 Reisinger, J. and R. J. Mooney (2010). Multi-prototype vector-space models of word mean-  
12170 ing. In *Proceedings of the North American Chapter of the Association for Computational Lin-*  
12171 *guistics (NAACL)*, pp. 109–117.
- 12172 Reiter, E. and R. Dale (2000). *Building natural language generation systems*. Cambridge  
12173 university press.
- 12174 Resnik, P., M. B. Olsen, and M. Diab (1999). The bible as a parallel corpus: Annotating the  
12175 ‘book of 2000 tongues’. *Computers and the Humanities* 33(1-2), 129–153.
- 12176 Resnik, P. and N. A. Smith (2003). The web as a parallel corpus. *Computational Linguis-*  
12177 *tics* 29(3), 349–380.
- 12178 Ribeiro, F. N., M. Araújo, P. Gonçalves, M. A. Gonçalves, and F. Benevenuto (2016).  
12179 Sentibench-a benchmark comparison of state-of-the-practice sentiment analysis meth-  
12180 ods. *EPJ Data Science* 5(1), 1–29.
- 12181 Richardson, M., C. J. Burges, and E. Renshaw (2013). MCTest: A challenge dataset for  
12182 the open-domain machine comprehension of text. In *Proceedings of Empirical Methods for*  
12183 *Natural Language Processing (EMNLP)*, pp. 193–203.
- 12184 Riedel, S., L. Yao, and A. McCallum (2010). Modeling relations and their mentions without  
12185 labeled text. In *Proceedings of the European Conference on Machine Learning and Principles*  
12186 *and Practice of Knowledge Discovery in Databases (ECML)*, pp. 148–163.
- 12187 Riedl, M. O. and R. M. Young (2010). Narrative planning: Balancing plot and character.  
12188 *Journal of Artificial Intelligence Research* 39, 217–268.
- 12189 Rieser, V. and O. Lemon (2011). *Reinforcement learning for adaptive dialogue systems: a data-*  
12190 *driven methodology for dialogue management and natural language generation*. Springer Sci-  
12191 ence & Business Media.

- 12192 Riloff, E. (1996). Automatically generating extraction patterns from untagged text. In  
 12193 *Proceedings of the national conference on artificial intelligence*, pp. 1044–1049.
- 12194 Riloff, E. and J. Wiebe (2003). Learning extraction patterns for subjective expressions. In  
 12195 *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pp.  
 12196 105–112. Association for Computational Linguistics.
- 12197 Ritchie, G. (2001). Current directions in computational humour. *Artificial Intelligence Re-*  
 12198 *view* 16(2), 119–135.
- 12199 Ritter, A., C. Cherry, and W. B. Dolan (2011). Data-driven response generation in social  
 12200 media. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.  
 12201 583–593.
- 12202 Roark, B., M. Saraclar, and M. Collins (2007). Discriminative $i_l$ / $n_l/i_l$ -gram language  
 12203 modeling. *Computer Speech & Language* 21(2), 373–392.
- 12204 Robert, C. and G. Casella (2013). *Monte Carlo statistical methods*. Springer Science & Busi-  
 12205 ness Media.
- 12206 Rosenfeld, R. (1996). A maximum entropy approach to adaptive statistical language mod-  
 12207 elling. *Computer Speech & Language* 10(3), 187–228.
- 12208 Ross, S., G. Gordon, and D. Bagnell (2011). A reduction of imitation learning and struc-  
 12209 tured prediction to no-regret online learning. In *Proceedings of Artificial Intelligence and*  
 12210 *Statistics (AISTATS)*, pp. 627–635.
- 12211 Roy, N., J. Pineau, and S. Thrun (2000). Spoken dialogue management using probabilistic  
 12212 reasoning. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 93–  
 12213 100.
- 12214 Rudnicky, A. and W. Xu (1999). An agenda-based dialog management architecture for  
 12215 spoken language systems. In *IEEE Automatic Speech Recognition and Understanding Work-  
 12216 shop*, Volume 13.
- 12217 Rush, A. M., S. Chopra, and J. Weston (2015). A neural attention model for abstractive sen-  
 12218 tence summarization. In *Proceedings of Empirical Methods for Natural Language Processing  
 12219 (EMNLP)*, pp. 379–389.
- 12220 Rush, A. M., D. Sontag, M. Collins, and T. Jaakkola (2010). On dual decomposition and  
 12221 linear programming relaxations for natural language processing. In *Proceedings of Em-  
 12222 pirical Methods for Natural Language Processing (EMNLP)*, pp. 1–11.
- 12223 Russell, S. J. and P. Norvig (2009). *Artificial intelligence: a modern approach* (3rd ed.). Prentice  
 12224 Hall.

- 12225 Rutherford, A., V. Demberg, and N. Xue (2017). A systematic study of neural discourse  
12226 models for implicit discourse relation. In *Proceedings of the European Chapter of the Asso-*  
12227 *ciation for Computational Linguistics (EACL)*, pp. 281–291.
- 12228 Rutherford, A. T. and N. Xue (2014). Discovering implicit discourse relations through  
12229 brown cluster pair representation and coreference patterns. In *Proceedings of the Euro-*  
12230 *pean Chapter of the Association for Computational Linguistics (EACL)*.
- 12231 Sag, I. A., T. Baldwin, F. Bond, A. Copestake, and D. Flickinger (2002). Multiword expres-  
12232 sions: A pain in the neck for nlp. In *International Conference on Intelligent Text Processing*  
12233 and *Computational Linguistics*, pp. 1–15. Springer.
- 12234 Sagae, K. (2009). Analysis of discourse structure with syntactic dependencies and data-  
12235 driven shift-reduce parsing. In *Proceedings of the 11th International Conference on Parsing*  
12236 *Technologies*, pp. 81–84.
- 12237 Santos, C. D. and B. Zadrozny (2014). Learning character-level representations for part-of-  
12238 speech tagging. In *Proceedings of the International Conference on Machine Learning (ICML)*,  
12239 pp. 1818–1826.
- 12240 Sato, M.-A. and S. Ishii (2000). On-line em algorithm for the normalized gaussian network.  
12241 *Neural computation* 12(2), 407–432.
- 12242 Saurí, R. and J. Pustejovsky (2009). Factbank: a corpus annotated with event factuality.  
12243 *Language resources and evaluation* 43(3), 227.
- 12244 Saxe, A. M., J. L. McClelland, and S. Ganguli (2014). Exact solutions to the nonlinear  
12245 dynamics of learning in deep linear neural networks. In *Proceedings of the International*  
12246 *Conference on Learning Representations (ICLR)*.
- 12247 Schank, R. C. and R. Abelson (1977). *Scripts, goals, plans, and understanding*. Hillsdale, NJ:  
12248 Erlbaum.
- 12249 Schapire, R. E. and Y. Singer (2000). Boostexter: A boosting-based system for text catego-  
12250 *Machine learning* 39(2-3), 135–168.
- 12251 Schaul, T., S. Zhang, and Y. LeCun (2013). No more pesky learning rates. In *Proceedings of*  
12252 *the International Conference on Machine Learning (ICML)*, pp. 343–351.
- 12253 Schnabel, T., I. Labutov, D. Mimno, and T. Joachims (2015). Evaluation methods for un-  
12254 supervised word embeddings. In *Proceedings of Empirical Methods for Natural Language*  
12255 *Processing (EMNLP)*, pp. 298–307.
- 12256 Schneider, N., J. Flanigan, and T. O’Gorman (2015). The logic of amr: Practical, unified,  
12257 graph-based sentence semantics for nlp. In *Proceedings of the North American Chapter of*  
12258 *the Association for Computational Linguistics (NAACL)*, pp. 4–5.

- 12259 Schütze, H. (1998). Automatic word sense discrimination. *Computational linguistics* 24(1),  
12260 97–123.
- 12261 Schwarm, S. E. and M. Ostendorf (2005). Reading level assessment using support vector  
12262 machines and statistical language models. In *Proceedings of the Association for Compu-*  
12263 *tational Linguistics (ACL)*, pp. 523–530.
- 12264 See, A., P. J. Liu, and C. D. Manning (2017). Get to the point: Summarization with pointer-  
12265 generator networks. In *Proceedings of the Association for Computational Linguistics (ACL)*,  
12266 pp. 1073–1083.
- 12267 Sennrich, R., B. Haddow, and A. Birch (2016). Neural machine translation of rare words  
12268 with subword units. In *Proceedings of the Association for Computational Linguistics (ACL)*,  
12269 pp. 1715–1725.
- 12270 Serban, I. V., A. Sordoni, Y. Bengio, A. C. Courville, and J. Pineau (2016). Building end-to-  
12271 end dialogue systems using generative hierarchical neural network models. In *Proceed-*  
12272 *ings of the National Conference on Artificial Intelligence (AAAI)*, pp. 3776–3784.
- 12273 Settles, B. (2012). Active learning. *Synthesis Lectures on Artificial Intelligence and Machine*  
12274 *Learning* 6(1), 1–114.
- 12275 Shang, L., Z. Lu, and H. Li (2015). Neural responding machine for short-text conversation.  
12276 In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1577–1586.
- 12277 Shen, D. and M. Lapata (2007). Using semantic roles to improve question answering. In  
12278 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 12–21.
- 12279 Shen, S., Y. Cheng, Z. He, W. He, H. Wu, M. Sun, and Y. Liu (2016). Minimum risk train-  
12280 ing for neural machine translation. In *Proceedings of the Association for Computation-*  
12281 *al Linguistics (ACL)*, pp. 1683–1692.
- 12282 Shen, W., J. Wang, and J. Han (2015). Entity linking with a knowledge base: Issues, tech-  
12283 niques, and solutions. *IEEE Transactions on Knowledge and Data Engineering* 27(2), 443–  
12284 460.
- 12285 Shieber, S. M. (1985). Evidence against the context-freeness of natural language. *Linguistics*  
12286 and Philosophy
- 12287 Siegelmann, H. T. and E. D. Sontag (1995). On the computational power of neural nets.  
12288 *Journal of computer and system sciences* 50(1), 132–150.
- 12289 Singh, S., A. Subramanya, F. Pereira, and A. McCallum (2011). Large-scale cross-  
12290 document coreference using distributed inference and hierarchical models. In *Proceed-*  
12291 *ings of the Association for Computational Linguistics (ACL)*, pp. 793–803.

- 12292 Sipser, M. (2012). *Introduction to the Theory of Computation*. Cengage Learning.
- 12293 Smith, D. A. and J. Eisner (2006). Minimum risk annealing for training log-linear models.  
12294 In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 787–794.
- 12295 Smith, D. A. and J. Eisner (2008). Dependency parsing by belief propagation. In *Proceed-  
12296 ings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 145–156.
- 12297 Smith, D. A. and N. A. Smith (2007). Probabilistic models of nonprojective dependency  
12298 trees. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.  
12299 132–140.
- 12300 Smith, N. A. (2011). Linguistic structure prediction. *Synthesis Lectures on Human Language  
12301 Technologies* 4(2), 1–274.
- 12302 Snover, M., B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul (2006). A study of transla-  
12303 tion edit rate with targeted human annotation. In *Proceedings of association for machine  
12304 translation in the Americas*, Volume 200.
- 12305 Snow, R., B. O'Connor, D. Jurafsky, and A. Y. Ng (2008). Cheap and fast—but is it good?:  
12306 evaluating non-expert annotations for natural language tasks. In *Proceedings of Empirical  
12307 Methods for Natural Language Processing (EMNLP)*, pp. 254–263.
- 12308 Snyder, B. and R. Barzilay (2007). Database-text alignment via structured multilabel classi-  
12309 fication. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*,  
12310 pp. 1713–1718.
- 12311 Socher, R., J. Bauer, C. D. Manning, and A. Y. Ng (2013). Parsing with compositional vector  
12312 grammars. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 12313 Socher, R., B. Huval, C. D. Manning, and A. Y. Ng (2012). Semantic compositionality  
12314 through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Em-  
12315 pirical Methods in Natural Language Processing and Computational Natural Language Learn-  
12316 ing*, pp. 1201–1211. Association for Computational Linguistics.
- 12317 Socher, R., A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts (2013).  
12318 Recursive deep models for semantic compositionality over a sentiment treebank. In  
12319 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12320 Søgaard, A. (2013). Semi-supervised learning and domain adaptation in natural language  
12321 processing. *Synthesis Lectures on Human Language Technologies* 6(2), 1–103.
- 12322 Solorio, T. and Y. Liu (2008). Learning to predict code-switching points. In *Proceedings  
12323 of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 973–981. Association  
12324 for Computational Linguistics.

- 12325 Somasundaran, S., G. Namata, J. Wiebe, and L. Getoor (2009). Supervised and unsupervised methods in employing discourse relations for improving opinion polarity classification. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12328 Somasundaran, S. and J. Wiebe (2009). Recognizing stances in online debates. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 226–234.
- 12330 Song, L., B. Boots, S. M. Siddiqi, G. J. Gordon, and A. J. Smola (2010). Hilbert space embeddings of hidden markov models. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 991–998.
- 12333 Song, L., Y. Zhang, X. Peng, Z. Wang, and D. Gildea (2016). Amr-to-text generation as a traveling salesman problem. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 2084–2089.
- 12336 Soon, W. M., H. T. Ng, and D. C. Y. Lim (2001). A machine learning approach to coreference resolution of noun phrases. *Computational linguistics* 27(4), 521–544.
- 12338 Sordoni, A., M. Galley, M. Auli, C. Brockett, Y. Ji, M. Mitchell, J.-Y. Nie, J. Gao, and B. Dolan (2015). A neural network approach to context-sensitive generation of conversational responses. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.
- 12342 Soriceut, R. and D. Marcu (2003). Sentence level discourse parsing using syntactic and lexical information. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 149–156.
- 12345 Sowa, J. F. (2000). *Knowledge representation: logical, philosophical, and computational foundations*. Pacific Grove, CA: Brooks/Cole.
- 12347 Spärck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation* 28(1), 11–21.
- 12349 Spitkovsky, V. I., H. Alshawi, D. Jurafsky, and C. D. Manning (2010). Viterbi training improves unsupervised dependency parsing. In *CONLL*, pp. 9–17.
- 12351 Sporleder, C. and M. Lapata (2005). Discourse chunking and its application to sentence compression. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 257–264.
- 12354 Sproat, R., A. Black, S. Chen, S. Kumar, M. Ostendorf, and C. Richards (2001). Normalization of non-standard words. *Computer Speech & Language* 15(3), 287–333.
- 12356 Sproat, R., W. Gale, C. Shih, and N. Chang (1996). A stochastic finite-state word-segmentation algorithm for chinese. *Computational linguistics* 22(3), 377–404.

- 12358 Sra, S., S. Nowozin, and S. J. Wright (2012). *Optimization for machine learning*. MIT Press.
- 12359 Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014).
- 12360 Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1), 1929–1958.
- 12361
- 12362 Srivastava, R. K., K. Greff, and J. Schmidhuber (2015). Training very deep networks. In *Neural Information Processing Systems (NIPS)*, pp. 2377–2385.
- 12363
- 12364 Stab, C. and I. Gurevych (2014a). Annotating argument components and relations in per-  
12365 suasive essays. In *Proceedings of the International Conference on Computational Linguistics*  
12366 (COLING), pp. 1501–1510.
- 12367 Stab, C. and I. Gurevych (2014b). Identifying argumentative discourse structures in per-  
12368 suasive essays. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Lan-  
12369 guage Processing (EMNLP)*, pp. 46–56.
- 12370 Stede, M. (2011, nov). *Discourse Processing*, Volume 4 of *Synthesis Lectures on Human Lan-  
12371 guage Technologies*. Morgan & Claypool Publishers.
- 12372 Steedman, M. and J. Baldridge (2011). Combinatory categorial grammar. In *Non-  
12373 Transformational Syntax: Formal and Explicit Models of Grammar*. Wiley-Blackwell.
- 12374 Stenetorp, P., S. Pyysalo, G. Topić, T. Ohta, S. Ananiadou, and J. Tsujii (2012). Brat: a web-  
12375 based tool for nlp-assisted text annotation. In *Proceedings of the European Chapter of the  
12376 Association for Computational Linguistics (EACL)*, pp. 102–107.
- 12377 Stern, M., J. Andreas, and D. Klein (2017). A minimal span-based neural constituency  
12378 parser. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 12379 Stolcke, A., K. Ries, N. Coccaro, E. Shriberg, R. Bates, D. Jurafsky, P. Taylor, R. Martin,  
12380 C. Van Ess-Dykema, and M. Meteer (2000). Dialogue act modeling for automatic tag-  
12381 ging and recognition of conversational speech. *Computational linguistics* 26(3), 339–373.
- 12382 Stone, P. J. (1966). *The General Inquirer: A Computer Approach to Content Analysis*. The MIT  
12383 Press.
- 12384 Stoyanov, V., N. Gilbert, C. Cardie, and E. Riloff (2009). Conundrums in noun phrase  
12385 coreference resolution: Making sense of the state-of-the-art. In *Proceedings of the Associa-  
12386 tion for Computational Linguistics (ACL)*, pp. 656–664.
- 12387 Strang, G. (2016). *Introduction to linear algebra* (Fifth ed.). Wellesley, MA: Wellesley-  
12388 Cambridge Press.

- 12389 Strubell, E., P. Verga, D. Belanger, and A. McCallum (2017). Fast and accurate entity recognition  
 12390 with iterated dilated convolutions. In *Proceedings of Empirical Methods for Natural  
 12391 Language Processing (EMNLP)*.
- 12392 Suchanek, F. M., G. Kasneci, and G. Weikum (2007). Yago: a core of semantic knowledge.  
 12393 In *Proceedings of the Conference on World-Wide Web (WWW)*, pp. 697–706.
- 12394 Sun, X., T. Matsuzaki, D. Okanohara, and J. Tsujii (2009). Latent variable perceptron algo-  
 12395 rithm for structured classification. In *Proceedings of the International Joint Conference on  
 12396 Artificial Intelligence (IJCAI)*, Volume 9, pp. 1236–1242.
- 12397 Sun, Y., L. Lin, D. Tang, N. Yang, Z. Ji, and X. Wang (2015). Modeling mention, context  
 12398 and entity with neural networks for entity disambiguation. In *IJCAI*, pp. 1333–1339.
- 12399 Sundermeyer, M., R. Schlüter, and H. Ney (2012). Lstm neural networks for language  
 12400 modeling. In *INTERSPEECH*.
- 12401 Surdeanu, M., J. Tibshirani, R. Nallapati, and C. D. Manning (2012). Multi-instance multi-  
 12402 label learning for relation extraction. In *Proceedings of Empirical Methods for Natural Lan-  
 12403 guage Processing (EMNLP)*, pp. 455–465.
- 12404 Sutskever, I., O. Vinyals, and Q. V. Le (2014). Sequence to sequence learning with neural  
 12405 networks. In *Neural Information Processing Systems (NIPS)*, pp. 3104–3112.
- 12406 Sutton, R. S. and A. G. Barto (1998). *Reinforcement learning: An introduction*, Volume 1. MIT  
 12407 press Cambridge.
- 12408 Sutton, R. S., D. A. McAllester, S. P. Singh, and Y. Mansour (2000). Policy gradient methods  
 12409 for reinforcement learning with function approximation. In *Neural Information Process-  
 12410 ing Systems (NIPS)*, pp. 1057–1063.
- 12411 Taboada, M., J. Brooke, M. Tofiloski, K. Voll, and M. Stede (2011). Lexicon-based methods  
 12412 for sentiment analysis. *Computational linguistics* 37(2), 267–307.
- 12413 Taboada, M. and W. C. Mann (2006). Rhetorical structure theory: Looking back and mov-  
 12414 ing ahead. *Discourse studies* 8(3), 423–459.
- 12415 Täckström, O., K. Ganchev, and D. Das (2015). Efficient inference and structured learning  
 12416 for semantic role labeling. *Transactions of the Association for Computational Linguistics* 3,  
 12417 29–41.
- 12418 Täckström, O., R. McDonald, and J. Uszkoreit (2012). Cross-lingual word clusters for  
 12419 direct transfer of linguistic structure. In *Proceedings of the North American Chapter of the  
 12420 Association for Computational Linguistics (NAACL)*, pp. 477–487.

- 12421 Tang, D., B. Qin, and T. Liu (2015). Document modeling with gated recurrent neural net-  
12422 work for sentiment classification. In *Proceedings of Empirical Methods for Natural Language  
12423 Processing (EMNLP)*, pp. 1422–1432.
- 12424 Taskar, B., C. Guestrin, and D. Koller (2003). Max-margin markov networks. In *Neural  
12425 Information Processing Systems (NIPS)*.
- 12426 Tausczik, Y. R. and J. W. Pennebaker (2010). The psychological meaning of words: LIWC  
12427 and computerized text analysis methods. *Journal of Language and Social Psychology* 29(1),  
12428 24–54.
- 12429 Teh, Y. W. (2006). A hierarchical bayesian language model based on pitman-yor processes.  
12430 In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 985–992.
- 12431 Tesnière, L. (1966). *Éléments de syntaxe structurale* (second ed.). Paris: Klincksieck.
- 12432 Teufel, S., J. Carletta, and M. Moens (1999). An annotation scheme for discourse-level  
12433 argumentation in research articles. In *Proceedings of the European Chapter of the Association  
12434 for Computational Linguistics (EACL)*, pp. 110–117.
- 12435 Teufel, S. and M. Moens (2002). Summarizing scientific articles: experiments with rele-  
12436 vance and rhetorical status. *Computational linguistics* 28(4), 409–445.
- 12437 Thomas, M., B. Pang, and L. Lee (2006). Get out the vote: Determining support or opposi-  
12438 tion from Congressional floor-debate transcripts. In *Proceedings of Empirical Methods for  
12439 Natural Language Processing (EMNLP)*, pp. 327–335.
- 12440 Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal  
12441 Statistical Society. Series B (Methodological)*, 267–288.
- 12442 Titov, I. and J. Henderson (2007). Constituent parsing with incremental sigmoid belief  
12443 networks. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 632–  
12444 639.
- 12445 Toutanova, K., D. Klein, C. D. Manning, and Y. Singer (2003). Feature-rich part-of-speech  
12446 tagging with a cyclic dependency network. In *Proceedings of the North American Chapter  
12447 of the Association for Computational Linguistics (NAACL)*.
- 12448 Trivedi, R. and J. Eisenstein (2013). Discourse connectors for latent subjectivity in senti-  
12449 ment analysis. In *Proceedings of the North American Chapter of the Association for Compu-  
12450 tational Linguistics (NAACL)*, pp. 808–813.
- 12451 Tromble, R. W. and J. Eisner (2006). A fast finite-state relaxation method for enforcing  
12452 global constraints on sequence decoding. In *Proceedings of the North American Chapter of  
12453 the Association for Computational Linguistics (NAACL)*, pp. 423.

- 12454 Tsochantaridis, I., T. Hofmann, T. Joachims, and Y. Altun (2004). Support vector machine  
12455 learning for interdependent and structured output spaces. In *Proceedings of the twenty-*  
12456 *first international conference on Machine learning*, pp. 104. ACM.
- 12457 Tsvetkov, Y., M. Faruqui, W. Ling, G. Lample, and C. Dyer (2015). Evaluation of word  
12458 vector representations by subspace alignment. In *Proceedings of Empirical Methods for*  
12459 *Natural Language Processing (EMNLP)*, pp. 2049–2054.
- 12460 Tu, Z., Z. Lu, Y. Liu, X. Liu, and H. Li (2016). Modeling coverage for neural machine  
12461 translation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 76–  
12462 85.
- 12463 Turian, J., L. Ratinov, and Y. Bengio (2010). Word representations: a simple and general  
12464 method for semi-supervised learning. In *Proceedings of the Association for Computational*  
12465 *Linguistics (ACL)*, pp. 384–394.
- 12466 Turing, A. M. (2009). Computing machinery and intelligence. In *Parsing the Turing Test*,  
12467 pp. 23–65. Springer.
- 12468 Turney, P. D. and P. Pantel (2010). From frequency to meaning: Vector space models of  
12469 semantics. *Journal of Artificial Intelligence Research* 37, 141–188.
- 12470 Tutin, A. and R. Kittredge (1992). Lexical choice in context: generating procedural texts.  
12471 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.  
12472 763–769.
- 12473 Twain, M. (1997). *A Tramp Abroad*. New York: Penguin.
- 12474 Tzeng, E., J. Hoffman, T. Darrell, and K. Saenko (2015). Simultaneous deep transfer across  
12475 domains and tasks. In *Proceedings of the IEEE International Conference on Computer Vision*,  
12476 pp. 4068–4076.
- 12477 Usunier, N., D. Buffoni, and P. Gallinari (2009). Ranking with ordered weighted pairwise  
12478 classification. In *Proceedings of the International Conference on Machine Learning (ICML)*,  
12479 pp. 1057–1064.
- 12480 Uthus, D. C. and D. W. Aha (2013). The ubuntu chat corpus for multiparicipant chat  
12481 analysis. In *AAAI Spring Symposium: Analyzing Microtext*, Volume 13, pp. 01.
- 12482 Utiyama, M. and H. Isahara (2001). A statistical model for domain-independent text seg-  
12483 mentation. In *Proceedings of the 39th Annual Meeting on Association for Computational*  
12484 *Linguistics*, pp. 499–506. Association for Computational Linguistics.

- 12485 Utiyama, M. and H. Isahara (2007). A comparison of pivot methods for phrase-based  
12486 statistical machine translation. In *Human Language Technologies 2007: The Conference of  
12487 the North American Chapter of the Association for Computational Linguistics; Proceedings of  
12488 the Main Conference*, pp. 484–491.
- 12489 Uzuner, Ö., X. Zhang, and T. Sibanda (2009). Machine learning and rule-based approaches  
12490 to assertion classification. *Journal of the American Medical Informatics Association* 16(1),  
12491 109–115.
- 12492 Vadas, D. and J. R. Curran (2011). Parsing noun phrases in the penn treebank. *Computa-  
12493 tional Linguistics* 37(4), 753–809.
- 12494 Van Eynde, F. (2006). NP-internal agreement and the structure of the noun phrase. *Journal  
12495 of Linguistics* 42(1), 139–186.
- 12496 Van Gael, J., A. Vlachos, and Z. Ghahramani (2009). The infinite hmm for unsuper-  
12497 vised pos tagging. In *Proceedings of Empirical Methods for Natural Language Processing  
12498 (EMNLP)*, pp. 678–687.
- 12499 Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and  
12500 I. Polosukhin (2017). Attention is all you need. In *Neural Information Processing Systems  
12501 (NIPS)*, pp. 6000–6010.
- 12502 Velldal, E., L. Øvrelid, J. Read, and S. Oopen (2012). Speculation and negation: Rules,  
12503 rankers, and the role of syntax. *Computational linguistics* 38(2), 369–410.
- 12504 Versley, Y. (2011). Towards finer-grained tagging of discourse connectives. In *Proceedings  
12505 of the Workshop Beyond Semantics: Corpus-based Investigations of Pragmatic and Discourse  
12506 Phenomena*, pp. 2–63.
- 12507 Vilain, M., J. Burger, J. Aberdeen, D. Connolly, and L. Hirschman (1995). A model-  
12508 theoretic coreference scoring scheme. In *Proceedings of the 6th conference on Message  
12509 understanding*, pp. 45–52. Association for Computational Linguistics.
- 12510 Vincent, P., H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol (2010). Stacked de-  
12511 noising autoencoders: Learning useful representations in a deep network with a local  
12512 denoising criterion. *Journal of Machine Learning Research* 11(Dec), 3371–3408.
- 12513 Vincze, V., G. Szarvas, R. Farkas, G. Móra, and J. Csirik (2008). The bioscope corpus:  
12514 biomedical texts annotated for uncertainty, negation and their scopes. *BMC bioinformat-  
12515 ics* 9(11), S9.
- 12516 Vinyals, O., A. Toshev, S. Bengio, and D. Erhan (2015). Show and tell: A neural image cap-  
12517 tion generator. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference  
12518 on*, pp. 3156–3164. IEEE.

- 12519 Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum  
 12520 decoding algorithm. *IEEE transactions on Information Theory* 13(2), 260–269.
- 12521 Voll, K. and M. Taboada (2007). Not all words are created equal: Extracting semantic  
 12522 orientation as a function of adjective relevance. In *Proceedings of Australian Conference  
 12523 on Artificial Intelligence*.
- 12524 Wager, S., S. Wang, and P. S. Liang (2013). Dropout training as adaptive regularization. In  
 12525 *Neural Information Processing Systems (NIPS)*, pp. 351–359.
- 12526 Wainwright, M. J. and M. I. Jordan (2008). Graphical models, exponential families, and  
 12527 variational inference. *Foundations and Trends® in Machine Learning* 1(1-2), 1–305.
- 12528 Walker, M. A. (2000). An application of reinforcement learning to dialogue strategy selec-  
 12529 tion in a spoken dialogue system for email. *Journal of Artificial Intelligence Research* 12,  
 12530 387–416.
- 12531 Walker, M. A., J. E. Cahn, and S. J. Whittaker (1997). Improvising linguistic style: Social  
 12532 and affective bases for agent personality. In *Proceedings of the first international conference  
 12533 on Autonomous agents*, pp. 96–105. ACM.
- 12534 Wang, C., N. Xue, and S. Pradhan (2015). A Transition-based Algorithm for AMR Parsing.  
 12535 In *Proceedings of the North American Chapter of the Association for Computational Linguistics  
 12536 (NAACL)*, pp. 366–375.
- 12537 Wang, H., T. Onishi, K. Gimpel, and D. McAllester (2017). Emergent predication structure  
 12538 in hidden state vectors of neural readers. In *Proceedings of the 2nd Workshop on Represen-  
 12539 tation Learning for NLP*, pp. 26–36.
- 12540 Weaver, W. (1955). Translation. *Machine translation of languages* 14, 15–23.
- 12541 Webber, B. (2004, sep). D-LTAG: extending lexicalized TAG to discourse. *Cognitive Sci-  
 12542 ence* 28(5), 751–779.
- 12543 Webber, B., M. Egg, and V. Kordoni (2012). Discourse structure and language technology.  
 12544 *Journal of Natural Language Engineering* 1.
- 12545 Webber, B. and A. Joshi (2012). Discourse structure and computation: past, present and  
 12546 future. In *Proceedings of the ACL-2012 Special Workshop on Rediscovering 50 Years of Dis-  
 12547 coveries*, pp. 42–54. Association for Computational Linguistics.
- 12548 Wei, G. C. and M. A. Tanner (1990). A monte carlo implementation of the em algorithm  
 12549 and the poor man’s data augmentation algorithms. *Journal of the American Statistical  
 12550 Association* 85(411), 699–704.

- 12551 Weinberger, K., A. Dasgupta, J. Langford, A. Smola, and J. Attenberg (2009). Feature  
12552 hashing for large scale multitask learning. In *Proceedings of the International Conference*  
12553 *on Machine Learning (ICML)*, pp. 1113–1120.
- 12554 Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language  
12555 communication between man and machine. *Communications of the ACM* 9(1), 36–45.
- 12556 Wellner, B. and J. Pustejovsky (2007). Automatically identifying the arguments of dis-  
12557 course connectives. In *Proceedings of Empirical Methods for Natural Language Processing*  
12558 (*EMNLP*), pp. 92–101.
- 12559 Wen, T.-H., M. Gasic, N. Mrkšić, P.-H. Su, D. Vandyke, and S. Young (2015). Semantically  
12560 conditioned lstm-based natural language generation for spoken dialogue systems. In  
12561 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1711–1721.
- 12562 Weston, J., S. Bengio, and N. Usunier (2011). Wsabie: Scaling up to large vocabulary image  
12563 annotation. In *IJCAI*, Volume 11, pp. 2764–2770.
- 12564 Wiebe, J., T. Wilson, and C. Cardie (2005). Annotating expressions of opinions and emo-  
12565 tions in language. *Language resources and evaluation* 39(2), 165–210.
- 12566 Wieting, J., M. Bansal, K. Gimpel, and K. Livescu (2015). Towards universal paraphrastic  
12567 sentence embeddings. *arXiv preprint arXiv:1511.08198*.
- 12568 Wieting, J., M. Bansal, K. Gimpel, and K. Livescu (2016). CHARAGRAM: Embedding  
12569 words and sentences via character n-grams. In *Proceedings of Empirical Methods for Nat-*  
12570 *ural Language Processing (EMNLP)*, pp. 1504–1515.
- 12571 Williams, J. D. and S. Young (2007). Partially observable markov decision processes for  
12572 spoken dialog systems. *Computer Speech & Language* 21(2), 393–422.
- 12573 Williams, P., R. Sennrich, M. Post, and P. Koehn (2016). Syntax-based statistical machine  
12574 translation. *Synthesis Lectures on Human Language Technologies* 9(4), 1–208.
- 12575 Wilson, T., J. Wiebe, and P. Hoffmann (2005). Recognizing contextual polarity in phrase-  
12576 level sentiment analysis. In *Proceedings of Empirical Methods for Natural Language Pro-*  
12577 *cessing (EMNLP)*, pp. 347–354.
- 12578 Winograd, T. (1972). Understanding natural language. *Cognitive psychology* 3(1), 1–191.
- 12579 Wiseman, S., A. M. Rush, and S. M. Shieber (2016). Learning global features for corefer-  
12580 ence resolution. In *Proceedings of the North American Chapter of the Association for Compu-*  
12581 *tational Linguistics (NAACL)*, pp. 994–1004.

- 12582 Wiseman, S., S. Shieber, and A. Rush (2017). Challenges in data-to-document generation.  
 12583 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 2253–  
 12584 2263.
- 12585 Wiseman, S. J., A. M. Rush, S. M. Shieber, and J. Weston (2015). Learning anaphoricity and  
 12586 antecedent ranking features for coreference resolution. In *Proceedings of the Association*  
 12587 for Computational Linguistics (ACL).
- 12588 Wolf, F. and E. Gibson (2005). Representing discourse coherence: A corpus-based study.  
 12589 *Computational Linguistics* 31(2), 249–287.
- 12590 Wolfe, T., M. Dredze, and B. Van Durme (2017). Pocket knowledge base population. In  
 12591 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 305–310.
- 12592 Wong, Y. W. and R. Mooney (2007). Generation by inverting a semantic parser that uses  
 12593 statistical machine translation. In *Proceedings of the North American Chapter of the Associa-*  
 12594 *tion for Computational Linguistics (NAACL)*, pp. 172–179.
- 12595 Wong, Y. W. and R. J. Mooney (2006). Learning for semantic parsing with statistical ma-  
 12596 chine translation. In *Proceedings of the North American Chapter of the Association for Com-*  
 12597 *putational Linguistics (NAACL)*, pp. 439–446.
- 12598 Wu, B. Y. and K.-M. Chao (2004). *Spanning trees and optimization problems*. CRC Press.
- 12599 Wu, D. (1997). Stochastic inversion transduction grammars and bilingual parsing of par-  
 12600 allel corpora. *Computational linguistics* 23(3), 377–403.
- 12601 Wu, F. and D. S. Weld (2010). Open information extraction using wikipedia. In *Proceedings*  
 12602 of the Association for Computational Linguistics (ACL), pp. 118–127.
- 12603 Wu, X., R. Ward, and L. Bottou (2018). Wngrad: Learn the learning rate in gradient de-  
 12604 scent. *arXiv preprint arXiv:1803.02865*.
- 12605 Wu, Y., M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao,  
 12606 Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws,  
 12607 Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young,  
 12608 J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean (2016).  
 12609 Google’s neural machine translation system: Bridging the gap between human and ma-  
 12610 chine translation. *CoRR abs/1609.08144*.
- 12611 Xia, F. (2000). The part-of-speech tagging guidelines for the penn chinese treebank (3.0).  
 12612 Technical report, University of Pennsylvania Institute for Research in Cognitive Science.
- 12613 Xu, K., J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio  
 12614 (2015). Show, attend and tell: Neural image caption generation with visual attention.  
 12615 In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 2048–2057.

- 12616 Xu, W., X. Liu, and Y. Gong (2003). Document clustering based on non-negative matrix  
12617 factorization. In *SIGIR*, pp. 267–273. ACM.
- 12618 Xu, Y., L. Mou, G. Li, Y. Chen, H. Peng, and Z. Jin (2015). Classifying relations via long  
12619 short term memory networks along shortest dependency paths. In *Proceedings of Empirical  
12620 Methods for Natural Language Processing (EMNLP)*, pp. 1785–1794.
- 12621 Xuan Bach, N., N. L. Minh, and A. Shimazu (2012). A reranking model for discourse seg-  
12622 mentation using subtree features. In *Proceedings of the Special Interest Group on Discourse  
12623 and Dialogue (SIGDIAL)*.
- 12624 Xue, N. et al. (2003). Chinese word segmentation as character tagging. *Computational  
12625 Linguistics and Chinese Language Processing* 8(1), 29–48.
- 12626 Xue, N., H. T. Ng, S. Pradhan, R. Prasad, C. Bryant, and A. T. Rutherford (2015). The  
12627 CoNLL-2015 shared task on shallow discourse parsing. In *Proceedings of the Conference  
12628 on Natural Language Learning (CoNLL)*.
- 12629 Xue, N., H. T. Ng, S. Pradhan, A. Rutherford, B. L. Webber, C. Wang, and H. Wang (2016).  
12630 Conll 2016 shared task on multilingual shallow discourse parsing. In *CoNLL Shared  
12631 Task*, pp. 1–19.
- 12632 Yamada, H. and Y. Matsumoto (2003). Statistical dependency analysis with support vector  
12633 machines. In *Proceedings of IWPT*, Volume 3, pp. 195–206.
- 12634 Yamada, K. and K. Knight (2001). A syntax-based statistical translation model. In *Proceed-  
12635 ings of the 39th Annual Meeting on Association for Computational Linguistics*, pp. 523–530.  
12636 Association for Computational Linguistics.
- 12637 Yang, B. and C. Cardie (2014). Context-aware learning for sentence-level sentiment anal-  
12638 ysis with posterior regularization. In *Proceedings of the Association for Computational Lin-  
12639 guistics (ACL)*.
- 12640 Yang, Y., M.-W. Chang, and J. Eisenstein (2016). Toward socially-infused information ex-  
12641 traction: Embedding authors, mentions, and entities. In *Proceedings of Empirical Methods  
12642 for Natural Language Processing (EMNLP)*.
- 12643 Yang, Y. and J. Eisenstein (2013). A log-linear model for unsupervised text normalization.  
12644 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12645 Yang, Y. and J. Eisenstein (2015). Unsupervised multi-domain adaptation with feature em-  
12646 beddings. In *Proceedings of the North American Chapter of the Association for Computational  
12647 Linguistics (NAACL)*.

- 12648 Yang, Y., W.-t. Yih, and C. Meek (2015). WikiQA: A challenge dataset for open-domain  
 12649 question answering. In *Proceedings of Empirical Methods for Natural Language Processing*  
 12650 (*EMNLP*), pp. 2013–2018.
- 12651 Yannakoudakis, H., T. Briscoe, and B. Medlock (2011). A new dataset and method for  
 12652 automatically grading esol texts. In *Proceedings of the 49th Annual Meeting of the Associa-*  
 12653 *tion for Computational Linguistics: Human Language Technologies-Volume 1*, pp. 180–189.  
 12654 Association for Computational Linguistics.
- 12655 Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised meth-  
 12656 ods. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 189–196.  
 12657 Association for Computational Linguistics.
- 12658 Yee, L. C. and T. Y. Jones (2012, March). Apple ceo in china mission to clear up problems.  
 12659 *Reuters*. retrieved on March 26, 2017.
- 12660 Yi, Y., C.-Y. Lai, S. Petrov, and K. Keutzer (2011, October). Efficient parallel cky parsing on  
 12661 gpus. In *Proceedings of the 12th International Conference on Parsing Technologies*, Dublin,  
 12662 Ireland, pp. 175–185. Association for Computational Linguistics.
- 12663 Yu, C.-N. J. and T. Joachims (2009). Learning structural svms with latent variables. In  
 12664 *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1169–1176.
- 12665 Yu, F. and V. Koltun (2016). Multi-scale context aggregation by dilated convolutions. In  
 12666 *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 12667 Zaidan, O. F. and C. Callison-Burch (2011). Crowdsourcing translation: Professional qual-  
 12668 ity from non-professionals. In *Proceedings of the Association for Computational Linguistics*  
 12669 (*ACL*), pp. 1220–1229.
- 12670 Zaremba, W., I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv*  
 12671 preprint *arXiv:1409.2329*.
- 12672 Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint*  
 12673 *arXiv:1212.5701*.
- 12674 Zelenko, D., C. Aone, and A. Richardella (2003). Kernel methods for relation extraction.  
 12675 *The Journal of Machine Learning Research* 3, 1083–1106.
- 12676 Zelle, J. M. and R. J. Mooney (1996). Learning to parse database queries using induc-  
 12677 tive logic programming. In *Proceedings of the National Conference on Artificial Intelligence*  
 12678 (*AAAI*), pp. 1050–1055.
- 12679 Zeng, D., K. Liu, S. Lai, G. Zhou, and J. Zhao (2014). Relation classification via convolu-  
 12680 tional deep neural network. In *Proceedings of the International Conference on Computational*  
 12681 *Linguistics (COLING)*, pp. 2335–2344.

- 12682 Zettlemoyer, L. S. and M. Collins (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of UAI*.
- 12683
- 12684 Zhang, X., J. Zhao, and Y. LeCun (2015). Character-level convolutional networks for text classification. In *Neural Information Processing Systems (NIPS)*, pp. 649–657.
- 12685
- 12686 Zhang, Y. and S. Clark (2008). A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 562–571.
- 12687
- 12688
- 12689 Zhang, Y., T. Lei, R. Barzilay, T. Jaakkola, and A. Globerson (2014). Steps to excellence: Simple inference with refined scoring of dependency trees. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 197–207.
- 12690
- 12691
- 12692 Zhang, Y. and J. Nivre (2011). Transition-based dependency parsing with rich non-local features. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 188–193.
- 12693
- 12694 Zhou, J. and W. Xu (2015). End-to-end learning of semantic role labeling using recurrent neural networks. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1127–1137.
- 12695
- 12696
- 12697 Zhu, J., Z. Nie, X. Liu, B. Zhang, and J.-R. Wen (2009). Statsnowball: a statistical approach to extracting entity relationships. In *Proceedings of the Conference on World-Wide Web (WWW)*, pp. 101–110.
- 12698
- 12699
- 12700 Zhu, X., Z. Ghahramani, and J. D. Lafferty (2003). Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 912–919.
- 12701
- 12702
- 12703 Zhu, X. and A. B. Goldberg (2009). Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning* 3(1), 1–130.
- 12704
- 12705 Zipf, G. K. (1949). Human behavior and the principle of least effort.
- 12706
- 12707 Zirn, C., M. Niepert, H. Stuckenschmidt, and M. Strube (2011). Fine-grained sentiment analysis with structural features. In *IJCNLP*, Chiang Mai, Thailand, pp. 336–344.
- 12708
- 12709 Zou, W. Y., R. Socher, D. Cer, and C. D. Manning (2013). Bilingual word embeddings for phrase-based machine translation. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1393–1398.
- 12710



12711

# Index

- 12712    *K*-means, 108  
12713     $\alpha$ -conversion, 300  
12714     $\beta$ -conversion, 297  
12715     $\beta$ -reduction, 297  
12716    *n*-gram language models, 208  
12717    *n*-gram, 40  
12718    *F*-MEASURE, 94  
12719    BLEU, 439  
12720    WordNet, 86
- 12721    ablation test, 96  
12722    absolute discounting, 142  
12723    Abstract Meaning Representation  
             (AMR), 313, 325  
12724    abstractive summarization, 401, 468  
12725    accepting path, 203  
12727    accuracy, 38, 93  
12728    action (reinforcement learning), 375  
12729    active learning, 130  
12730    AdaDelta (online optimization), 75  
12731    AdaGrad (online optimization), 55, 75  
12732    Adam (online optimization), 75  
12733    adequacy (translation), 439  
12734    adjectives, 189  
12735    adjuncts (semantics), 312, 316  
12736    adpositions, 190  
12737    adverbs, 189  
12738    adversarial networks, 125  
12739    affix (morphology), 205  
12740    agenda-based dialogue systems, 472  
12741    agenda-based parsing, 284
- 12742    agent (thematic role), 314  
12743    alignment, 327  
12744    alignment (machine translation), 438,  
             443  
12746    alignment (text generation), 463  
12747    Amazon Mechanical Turk, 102  
12748    ambiguity, 220, 228  
12749    ambiguity, attachment, 239  
12750    ambiguity, complement structure, 239  
12751    ambiguity, coordination scope, 239  
12752    ambiguity, modifier scope, 239  
12753    ambiguity, particle versus preposition,  
             239  
12755    anaphoric, 370  
12756    anchored productions, 242  
12757    animacy (semantics), 313  
12758    annealing, 458  
12759    antecedent (coreference), 361, 369  
12760    antonymy, 86  
12761    apophony, 204  
12762    arc-eager dependency parsing, 275, 277  
12763    arc-factored dependency parsing, 269  
12764    arc-standard dependency parsing, 275  
12765    area under the curve (AUC), 95  
12766    argumentation, 400  
12767    argumentation mining, 401  
12768    arguments, 409  
12769    article (syntax), 194  
12770    aspect, 189  
12771    aspect-based opinion mining, 83  
12772    attachment ambiguity, 263

- 12773 attention mechanism, 381, 432, 451, 464  
 12774 autoencoder, 355  
 12775 autoencoder, denoising, 355, 414  
 12776 automated theorem provers, 292  
 12777 automatic differentiation, 70  
 12778 auxiliary verbs, 190  
 12779 average mutual information, 343  
 12780 averaged perceptron, 42  
 12781 backchannel, 199  
 12782 backoff, 142  
 12783 backpropagation, 69, 148  
 12784 backpropagation through time, 148  
 12785 backward recurrence, 177, 178  
 12786 backward-looking center, 390  
 12787 bag of words, 29  
 12788 balanced *F*-MEASURE, 95  
 12789 balanced test set, 93  
 12790 batch learning, 41  
 12791 batch normalization, 74  
 12792 batch optimization, 53  
 12793 Baum-Welch algorithm, 182  
 12794 Bayes' rule, 480  
 12795 Bayesian nonparametrics, 115, 258  
 12796 beam sampling, 184  
 12797 beam search, 277, 374, 455, 456  
 12798 Bell number, 372  
 12799 best-path algorithm, 208  
 12800 bias (learning theory), 38, 139  
 12801 bias-variance tradeoff, 38, 141  
 12802 biconvexity, 114  
 12803 bidirectional LSTM, 451  
 12804 bidirectional recurrent neural network,  
     180  
 12805 bigrams, 40, 82  
 12806 bilexical, 257  
 12808 bilexical features, 272  
 12809 bilinear product, 340  
 12810 binarization (context-free grammar),  
     221, 238  
 12812 binomial distribution, 96  
 12813 binomial random variable, 484  
 12814 binomial test, 96  
 12815 BIO notation, 196, 323  
 12816 biomedical natural language processing,  
     195  
 12818 bipartite graph, 329  
 12819 Bonferroni correction, 99  
 12820 boolean semiring, 209  
 12821 boosting, 62  
 12822 bootstrap samples, 98  
 12823 brevity penalty (machine translation),  
     440  
 12825 Brown clusters, 337, 342  
 12826 byte-pair encoding, 353, 454  
 12827 c-command, 363  
 12828 case marking, 194, 229  
 12829 Catalan number, 235  
 12830 cataphora, 362  
 12831 center embedding, 217  
 12832 centering theory, 365, 390  
 12833 chain FSA, 215  
 12834 chain rule of probability, 479  
 12835 chance agreement, 102  
 12836 character-level language models, 153  
 12837 chart parsing, 236  
 12838 chatbots, 474  
 12839 Chomsky Normal Form (CNF), 221  
 12840 Chu-Liu-Edmonds algorithm, 270  
 12841 CKY algorithm, 236  
 12842 class imbalance, 93  
 12843 classification weights, 29  
 12844 closed-vocabulary, 153  
 12845 closure (regular languages), 202  
 12846 cloze question answering, 431  
 12847 cluster ranking, 373  
 12848 clustering, 108  
 12849 co-training, 120  
 12850 coarse-to-fine attention, 467  
 12851 code switching, 192, 198  
 12852 Cohen's Kappa, 102

- 12853 coherence, 404  
 12854 cohesion, 387  
 12855 collapsed Gibbs sampling, 128  
 12856 collective entity linking, 414, 415  
 12857 collocation extraction, 354  
 12858 collocation features, 87  
 12859 combinatorial optimization, 19  
 12860 combinatory categorial grammar, 230  
 12861 complement clause, 223  
 12862 complement event (probability), 478  
 12863 composition (CCG), 231  
 12864 compositional vector grammars, 399  
 12865 compositionality, 18, 21, 352  
 12866 computation graph, 62, 69  
 12867 computational linguistics (versus natural language processing), 13  
 12868 computational social science, 17  
 12869 concept (AMR), 325  
 12870 conditional independence, 166, 482  
 12871 conditional log-likelihood, 66  
 12872 conditional probability, 50, 479  
 12873 conditional probability distribution, 483  
 12874 conditional random field, 174  
 12875 confidence interval, 98  
 12876 configuration (transition-based parsing), 275  
 12877 connected (graph theory), 327  
 12878 consistency (logic), 295  
 12879 constants (logic), 290  
 12880 constituents, 222  
 12881 constrained optimization, 47, 321  
 12882 constraint-driven learning, 130  
 12883 constraints, 321  
 12884 content selection (text generation), 461  
 12885 content words, 190  
 12886 context vector (attentional neural translation), 452  
 12887 context-free grammars (CFGs), 218  
 12888 context-free languages, 217, 218  
 12889 context-sensitive languages, 228  
 12890 continuous bag-of-words (CBOW), 345  
 12891 contradiction, 356  
 12892 conversational turns, 199  
 12893 convex optimization, 53  
 12894 convexity, 44, 72, 305, 484, 487  
 12895 convolutional neural networks, 67, 76, 82, 153, 182, 197, 421  
 12896 cooperative principle, 361  
 12897 coordinate ascent, 114  
 12898 coordinating conjunctions, 190  
 12899 coordinating discourse relations, 397  
 12900 copula, 189, 227, 266  
 12901 coreference resolution, 357, 361  
 12902 coreferent, 361  
 12903 cosine similarity, 349, 388  
 12904 cost-augmented decoding, 49, 173  
 12905 coverage (summarization), 401  
 12906 coverage loss, 469  
 12907 critical point, 72, 487  
 12908 cross-document coreference resolution, 412  
 12909 cross-entropy, 66, 422  
 12910 cross-serial dependencies, 229  
 12911 cross-validation, 39  
 12912 crowdsourcing, 102  
 12913 cumulative probability distribution, 97  
 12914 dead neurons, 65  
 12915 decidability (logic), 295  
 12916 decision trees, 62  
 12917 deep learning, 61  
 12918 deep LSTM, 450  
 12919 definiteness, 195  
 12920 delta function, 37  
 12921 denotation (semantics), 290  
 12922 dependency grammar, 263  
 12923 dependency graph, 264  
 12924 dependency parse, 263  
 12925 dependency path, 87, 320, 419  
 12926 dependent, 264  
 12927 derivation (context-free languages), 219  
 12928 derivation (parsing), 274, 280

- 12934 derivation (semantic parsing), 298, 302  
 12935 derivational ambiguity, 232  
 12936 derivational morphology, 204  
 12937 determiner, 191  
 12938 determiner phrase, 225  
 12939 deterministic FSA, 204  
 12940 development set, 39, 93  
 12941 dialogue acts, 102, 199, 476  
 12942 dialogue management, 472  
 12943 dialogue systems, 137, 471  
 12944 digital humanities, 17, 81  
 12945 dilated convolution, 77, 197  
 12946 Dirichlet distribution, 127  
 12947 discounting (language models), 142  
 12948 discourse, 387  
 12949 discourse connectives, 393  
 12950 discourse depth, 402  
 12951 discourse depth tree, 403  
 12952 discourse parsing, 392  
 12953 discourse relations, 357, 392  
 12954 discourse segment, 387  
 12955 discourse sense classification, 394  
 12956 discourse unit, 397  
 12957 discrete random variable, 482  
 12958 discriminative learning, 40  
 12959 disjoint events (probability), 478  
 12960 distant supervision, 130, 424, 425  
 12961 distributed semantics, 336  
 12962 distributional hypothesis, 335, 336  
 12963 distributional semantics, 22, 336  
 12964 distributional statistics, 87, 257, 336  
 12965 document frequency, 413  
 12966 domain adaptation, 107, 123  
 12967 dropout, 71, 149  
 12968 dual decomposition, 323  
 12969 dynamic computation graphs, 70  
 12970 dynamic oracle, 282  
 12971 dynamic programming, 161  
 12972 dynamic semantics, 307, 392  
 12973 E-step (expectation-maximization), 111  
 12974 early stopping, 43, 75  
 12975 early update, 282  
 12976 easy-first parsing, 285  
 12977 edit distance, 211, 441  
 12978 effective count (language models), 141  
 12979 elementary discourse units, 397  
 12980 elementwise nonlinearity, 63  
 12981 Elman unit, 147  
 12982 ELMo (embeddings from language models), 351  
 12983 embedding, 179, 345  
 12985 emission features, 160  
 12986 emotion, 84  
 12987 empirical Bayes, 128  
 12988 empty string, 202  
 12989 encoder-decoder, 448  
 12990 encoder-decoder model, 355, 464  
 12991 ensemble, 324  
 12992 ensemble learning, 450  
 12993 ensemble methods, 62  
 12994 entailment, 295, 356  
 12995 entities, 409  
 12996 entity embeddings, 413  
 12997 entity grid, 391  
 12998 entity linking, 361, 409, 411, 422  
 12999 entropy, 57, 111  
 13000 estimation, 484  
 13001 EuroParl corpus, 441  
 13002 event, 427  
 13003 event (probability), 477  
 13004 event coreference, 427  
 13005 event detection, 427  
 13006 event semantics, 311  
 13007 events, 409  
 13008 evidentiality, 194, 429  
 13009 exchange clustering, 344  
 13010 expectation, 483  
 13011 expectation maximization, 109, 143  
 13012 expectation semiring, 217  
 13013 expectation-maximization, in machine translation, 445  
 13014

- 13015 explicit semantic analysis, 338  
13016 exploding gradients, 149  
13017 extra-propositional semantics, 429  
13018 extractive question-answering, 432  
13019 extractive summarization, 401  
13020 extrinsic evaluation, 151, 349
- 13021 factoid questions, 328  
13022 factoids, 430  
13023 factor graph, 175  
13024 factuality, 429  
13025 false discovery rate, 99  
13026 False negative, 93  
13027 False positive, 93  
13028 false positive, 481  
13029 false positive rate, 95, 480  
13030 feature co-adaptation, 71  
13031 feature function, 30, 39  
13032 feature hashing, 92  
13033 feature noising, 72  
13034 feature selection, 56  
13035 features, 18  
13036 feedforward neural network, 64  
13037 fine-tuned word embeddings, 351  
13038 finite state acceptor (FSA), 203  
13039 finite state automata, 203  
13040 finite state composition, 214  
13041 finite state transducers, 206, 211  
13042 first-order logic, 293  
13043 fluency (translation), 439  
13044 fluent, 137  
13045 focus, 326  
13046 formal language theory, 201  
13047 forward recurrence, 176  
13048 forward variable, 178  
13049 forward variables, 176  
13050 forward-backward algorithm, 177, 216,  
    249  
13052 forward-looking centers, 390  
13053 frame, 471  
13054 frame elements, 316
- 13055 FrameNet, 316  
13056 frames, 316  
13057 Frobenius norm, 71  
13058 function (first-order logic), 294  
13059 function words, 190  
13060 functional margin, 47  
13061 functional segmentation, 387, 389
- 13062 garden path sentence, 158  
13063 gate (neural networks), 66, 451  
13064 gazetteer, 419  
13065 gazetteers, 367  
13066 gazetteers, 196  
13067 generalization, 43  
13068 generalized linear models, 57  
13069 generative model, 33, 245  
13070 generative models, 373  
13071 generative process, 143  
13072 generic referents, 366  
13073 geometric margin, 47  
13074 Gibbs sampling, 127, 415  
13075 gloss, 137, 191, 439  
13076 government and binding theory, 363  
13077 gradient, 45  
13078 gradient clipping, 74  
13079 gradient descent, 53  
13080 Gram matrix, 420  
13081 grammar induction, 251  
13082 grammaticality, 404  
13083 graph-based dependency parsing, 269  
13084 graphical model, 166  
13085 graphics processing units (GPUs), 182,  
    197  
13086 grid search, 38
- 13087 Hamming cost, 173  
13089 Hansards corpus, 441  
13090 hanzi, 89  
13091 hard expectation-maximization, 114  
13092 head, 264, 419  
13093 head percolation rules, 254

- 13094 head rules, 263  
 13095 head word, 222, 263, 367  
 13096 head words, 254  
 13097 hedging, 429  
 13098 held-out data, 151  
 13099 Hessian matrix, 54  
 13100 hidden Markov models, 166  
 13101 hidden variable perceptron, 217  
 13102 hierarchical clustering, 342  
 13103 hierarchical recurrent network, 475  
 13104 hierarchical softmax, 147, 347  
 13105 hierarchical topic segmentation, 389  
 13106 highway network, 66  
 13107 hinge loss, 44  
 13108 holonymy, 87  
 13109 homonym, 85  
 13110 human computation, 103  
 13111 hypergraph, 400  
 13112 hypernymy, 87  
 13113 hyperparameter, 38  
 13114 hyponymy, 87  
 13115 illocutionary force, 199  
 13116 implicit discourse relations, 394  
 13117 importance sampling, 458  
 13118 importance score, 458  
 13119 incremental expectation maximization, 114  
 13120 incremental perceptron, 282, 374  
 13122 independent and identically distributed (IID), 32  
 13124 indicator function, 37  
 13125 indicator random variable, 482  
 13126 inference, 159  
 13127 inference (logic), 289  
 13128 inference rules, 292  
 13129 inflection point, 488  
 13130 inflectional affixes, 90  
 13131 inflectional morphology, 189, 204, 212  
 13132 information extraction, 409  
 13133 information retrieval, 17, 423  
 13134 initiative (dialogue systems), 472  
 13135 input word embeddings, 147  
 13136 inside recurrence, 245, 246, 250  
 13137 inside-outside algorithm, 249, 258  
 13138 instance (AMR), 325  
 13139 instance labels, 32  
 13140 integer linear program, 434, 470  
 13141 integer linear programming, 321, 372, 403, 415  
 13143 inter-annotator agreement, 102  
 13144 interjections, 189  
 13145 interlingua, 438  
 13146 interpolated  $n$ -gram language model, 209  
 13148 interpolation, 143  
 13149 interval algebra, 428  
 13150 intrinsic evaluation, 151, 349  
 13151 inverse document frequency, 413  
 13152 inverse relation (AMR), 326  
 13153 inversion (finite state), 213  
 13154 irrealis, 82  
 13155 Jeffreys-Perks law, 141  
 13156 Jensen's inequality, 111  
 13157 joint probabilities, 483  
 13158 joint probability, 32, 50  
 13159 Kalman smoother, 184  
 13160 Katz backoff, 142  
 13161 kernel function, 420  
 13162 kernel methods, 62  
 13163 kernel support vector machine, 62, 420  
 13164 Kleene star, 202  
 13165 knapsack problem, 403  
 13166 knowledge base, 409  
 13167 knowledge base population, 422  
 13168 L-BFGS, 54  
 13169 label bias problem, 281  
 13170 label propagation, 122, 132  
 13171 labeled dependencies, 265  
 13172 labeled precision, 240

- 13173 labeled recall, 240  
 13174 Lagrange multiplier, 489  
 13175 Lagrangian, 489  
 13176 Lagrangian dual, 489  
 13177 lambda calculus, 297  
 13178 lambda expressions, 297  
 13179 language model, 138  
 13180 language models, 16  
 13181 Laplace smoothing, 38, 141  
 13182 large margin classification, 46  
 13183 latent conditional random fields, 305  
 13184 latent Dirichlet allocation, 388  
 13185 latent semantic analysis, 338, 340  
 13186 latent variable, 110, 216, 304, 425, 438  
 13187 latent variable perceptron, 305, 370  
 13188 layer normalization, 75, 453  
 13189 leaky ReLU, 65  
 13190 learning to search, 263, 283, 376  
 13191 least squares, 84  
 13192 leave-one-out, 39  
 13193 lemma, 85  
 13194 lemma (lexical semantics), 212  
 13195 lemmatization, 90  
 13196 Levenshtein edit distance, 211  
 13197 lexical entry, 298  
 13198 lexical features, 61  
 13199 lexical semantics, 85  
 13200 lexical unit (frame semantics), 316  
 13201 lexicalization, 254  
 13202 lexicalization (text generation), 461  
 13203 lexicalized tree-adjoining grammar for discourse (D-LTAG), 393  
 13204 lexicon, 298  
 13206 lexicon (CCG), 231  
 13207 lexicon-based classification, 84  
 13208 lexicon-based sentiment analysis, 82  
 13209 Lidstone smoothing, 141  
 13210 light verb, 327  
 13211 likelihood, 480  
 13212 linear regression, 84  
 13213 linear separability, 41  
 13214 linearization, 471  
 13215 literal character, 202  
 13216 local minimum, 488  
 13217 local optimum, 114  
 13218 locally-normalized objective, 281  
 13219 log-bilinear language model, 353  
 13220 logistic function, 57  
 13221 logistic loss, 51  
 13222 logistic regression, 50, 57  
 13223 Long short-term memories, 147  
 13224 long short-term memory, 149  
 13225 long short-term memory (LSTM), 66, 193, 449  
 13227 lookup layer, 67, 147  
 13228 loss function, 43  
 13229 LSTM, 149  
 13230 LSTM-CRF, 181, 324  
 13231 machine learning, 14  
 13232 machine reading, 431  
 13233 machine translation, 137  
 13234 Macro F-MEASURE, 94  
 13235 macro-reading, 410  
 13236 margin, 41, 46  
 13237 marginal probability distribution, 483  
 13238 marginal relevance, 469  
 13239 marginalize, 479  
 13240 markable, 368  
 13241 Markov assumption, 166  
 13242 Markov blanket, 166  
 13243 Markov Chain Monte Carlo (MCMC), 115, 127, 184  
 13244 Markov decision process, 472  
 13246 Markov random fields, 174  
 13247 matrix-tree theorem, 274  
 13248 max-margin Markov network, 173  
 13249 max-product algorithm, 169  
 13250 maximum a posteriori, 38, 485  
 13251 maximum conditional likelihood, 50  
 13252 maximum entropy, 57  
 13253 maximum likelihood, 484

- 13254 maximum likelihood estimate, 36  
 13255 maximum likelihood estimation, 32  
 13256 maximum spanning tree, 270  
 13257 McNemar’s test, 96  
 13258 meaning representation, 289  
 13259 membership problem, 201  
 13260 memory cell (LSTM), 149  
 13261 mention (coreference resolution), 361  
 13262 mention (entity), 409  
 13263 mention (information extraction), 411  
 13264 mention ranking, 370  
 13265 mention-pair model, 369  
 13266 meronymy, 87  
 13267 meteor, 441  
 13268 method of moments, 128  
 13269 micro *F*-MEASURE, 94  
 13270 micro-reading, 410  
 13271 mildly context-sensitive languages, 229  
 13272 minibatch, 54  
 13273 minimization (FSA), 206  
 13274 minimum error-rate training (MERT),  
     457  
 13275 minimum risk training, 457  
 13277 mixed-initiative, 472  
 13278 modality, 429  
 13279 model, 19  
 13280 model builder, 295  
 13281 model checker, 295  
 13282 model-theoretic semantics, 290  
 13283 modeling (machine learning), 53  
 13284 modifier (dependency grammar), 264  
 13285 modus ponens, 292  
 13286 moment-matching, 57  
 13287 monomorphemic, 206  
 13288 morphemes, 17, 153, 205  
 13289 morphological analysis, 212  
 13290 morphological generation, 212  
 13291 morphological segmentation, 171  
 13292 morphology, 91, 170, 204, 352, 454  
 13293 morphosyntactic, 188  
 13294 morphosyntactic attributes, 192  
 13295 morphotactic, 205  
 13296 multi-document summarization, 470  
 13297 multi-view learning, 120  
 13298 multilayer perceptron, 64  
 13299 multinomial distribution, 34  
 13300 multinomial naïve Bayes, 34  
 13301 multiple instance learning, 130, 425  
 13302 multiple-choice question answering, 431  
 13303 multitask learning, 130  
 13304 Naïve Bayes, 33  
 13305 name dictionary, 412  
 13306 named entities, 195  
 13307 named entity linking, 411  
 13308 named entity recognition, 181, 409, 411  
 13309 named entity types, 411  
 13310 narrow convolution, 77  
 13311 nearest-neighbor, 62, 420  
 13312 negation, 82, 429  
 13313 negative sampling, 347, 348, 417  
 13314 Neo-Davidsonian event semantics, 312  
 13315 neural attention, 450  
 13316 neural machine translation, 438  
 13317 neural networks, 61, 146  
 13318 NIL entity, 411  
 13319 noise-contrastive estimation, 147  
 13320 noisy channel model, 138, 442  
 13321 nominal modifier, 225  
 13322 nominals, 361  
 13323 nominals (coreference), 368  
 13324 non-core roles (AMR), 326  
 13325 non-terminals (context-free grammars),  
     219  
 13326 normalization, 90  
 13328 noun phrase, 14, 222  
 13329 nouns, 188  
 13330 NP-hard, 56, 415  
 13331 nuclearity (RST), 397  
 13332 nucleus (RST), 397  
 13333 null hypothesis, 96  
 13334 numeral (part of speech), 191

- 13335 numerical optimization, 20  
 13336 offset feature, 31  
 13337 one-dimensional convolution, 77  
 13338 one-hot, 382  
 13339 one-hot vector, 66  
 13340 one-tailed p-value, 97  
 13341 one-versus-all multiclass classification, 420  
 13342 one-versus-one multiclass classification, 420  
 13343 online expectation maximization, 114  
 13344 online learning, 41, 54  
 13345 ontology, 20  
 13346 open information extraction, 426  
 13347 open word classes, 188  
 13348 opinion polarity, 81  
 13349 oracle, 280, 328  
 13350 oracle (learning to search), 376  
 13351 orthography, 206, 214  
 13352 orthonormal matrix, 73  
 13353 out-of-vocabulary words, 193  
 13354 outside recurrence, 246, 250  
 13355 overfit, 38  
 13356 overfitting, 42  
 13357 overgeneration, 213, 223  
 13358 parallel corpora, 441  
 13359 parameters, 484  
 13360 paraphrase, 356  
 13361 parent annotation, 253  
 13362 parsing, 219  
 13363 part-of-speech, 187  
 13364 part-of-speech tagging, 157  
 13365 partially observable Markov decision process (POMDP), 474  
 13366 partially supervised learning, 258  
 13367 particle (part-of-speech), 191, 227  
 13368 partition, 479  
 13369 partition function, 176  
 13370 parts-of-speech, 17  
 13371 passive-aggressive, 489  
 13372 path (finite state automata), 203  
 13373 Penn Discourse Treebank (PDTB), 393  
 13374 Penn Treebank, 152, 171, 192, 222, 248  
 13375 perceptron, 41  
 13376 perplexity, 152  
 13377 phonology, 206  
 13378 phrase (syntax), 222  
 13379 phrase-structure grammar, 222  
 13380 pivot features, 124  
 13381 pointwise mutual information, 340  
 13382 policy, 375, 473  
 13383 policy (search), 282  
 13384 policy gradient, 376  
 13385 polysemous, 86  
 13386 pooling, 383  
 13387 pooling (convolution), 77, 383, 464  
 13388 positional encodings, 453  
 13389 positive pointwise mutual information, 341  
 13390 posterior, 480  
 13391 power law, 14  
 13392 pragmatics, 361  
 13393 pre-trained word representations, 351  
 13394 precision, 94, 480  
 13395 precision-at- $k$ , 95, 406  
 13396 precision-recall curve, 424  
 13397 precision-recall curves, 95  
 13398 predicate, 409  
 13399 predicative adjectives, 227  
 13400 predictive likelihood, 115  
 13401 prepositional phrase, 14, 227  
 13402 primal form, 489  
 13403 principle of compositionality, 296  
 13404 prior, 480  
 13405 prior expectation, 485  
 13406 probabilistic context-free grammars (PCFGs), 245  
 13407 probabilistic models, 484  
 13408 probabilistic topic model, 415  
 13409 probability density function, 483

- 13415 probability distribution, 482  
 13416 probability mass function, 97, 483  
 13417 probability simplex, 33  
 13418 processes, 428  
 13419 production rules, 219  
 13420 productivity, 205  
 13421 projection function, 124  
 13422 projectivity, 267  
 13423 pronominal anaphora resolution, 361  
 13424 pronoun, 190  
 13425 PropBank, 316  
 13426 proper nouns, 189  
 13427 property (logic), 293  
 13428 proposal distribution, 458  
 13429 proposition, 428  
 13430 propositions, 290, 291  
 13431 prosody, 199  
 13432 proto-roles, 315  
 13433 pseudo-projective dependency parsing, 277  
 13434 pumping lemma, 217  
 13435 pushdown automata, 219  
 13436 pushdown automaton, 260  
 13437 quadratic program, 47  
 13438 quantifier, 293  
 13439 quantifier, existential, 294  
 13440 quantifier, universal, 294  
 13441 quasi-Newton optimization, 54  
 13442 question answering, 355, 411  
 13443 random outcomes, 477  
 13444 ranking, 412  
 13445 ranking loss, 412  
 13446 recall, 94, 480  
 13447 recall-at- $k$ , 406  
 13448 receiver operating characteristic (ROC), 95  
 13449 rectified linear unit (ReLU), 65  
 13450 recurrent neural network, 147  
 13451 recurrent neural networks, 421  
 13452 recursion, 14  
 13453 recursive neural network, 403  
 13454 recursive neural networks, 260, 353, 356  
 13455 recursive production, 219  
 13456 reference arguments, 331  
 13457 reference resolution, 361  
 13458 reference translations, 439  
 13459 referent, 361  
 13460 referring expression, 389  
 13461 referring expressions, 361, 462  
 13462 reflexive pronoun, 363  
 13463 regression, 84  
 13464 regular expression, 202  
 13465 regular language, 202  
 13466 regularization, 49  
 13467 reification (events), 311  
 13468 reinforcement learning, 375, 456  
 13469 relation (logic), 299  
 13470 relation extraction, 283, 329, 417  
 13471 relations, 409  
 13472 relations (information extraction), 409  
 13473 relations (logic), 290  
 13474 relative frequency estimate, 36, 138, 485  
 13475 reranking, 259  
 13476 residual networks, 66  
 13477 retrofitting (word embeddings), 354  
 13478 Rhetorical Structure Theory (RST), 396  
 13479 rhetorical zones, 389  
 13480 RIBES (translation metric), 441  
 13481 ridge regression, 84  
 13482 risk, 457  
 13483 roll-in (reinforcement learning), 376  
 13484 roll-out (reinforcement learning), 377  
 13485 root (morpheme), 353  
 13486 saddle point, 488  
 13487 saddle points, 72  
 13488 sample space, 477  
 13489 satellite (RST), 397  
 13490 satisfaction (logic), 295  
 13491 scheduled sampling, 456

- |       |   |       |   |
|-------|---|-------|---|
| 13494 | schema, 409, 410, 426                                 | 13535 | slots (dialogue systems), 471                   |
| 13495 | search error, 261, 374                                | 13536 | smooth functions, 45                            |
| 13496 | second-order dependency parsing, 269                  | 13537 | smoothing, 38, 141                              |
| 13497 | second-order logic, 294                               | 13538 | soft $K$ -means, 109                            |
| 13498 | seed lexicon, 85                                      | 13539 | softmax, 63, 146, 422                           |
| 13499 | segmented discourse representation theory (SDRT), 392 | 13540 | source domain, 122                              |
| 13500 | self-attention, 452                                   | 13541 | source language, 437                            |
| 13501 | self-training, 120                                    | 13542 | spanning tree, 264                              |
| 13502 | semantic, 188   | 13543 | sparse matrix, 340                              |
| 13503 | semantic concordance, 88                              | 13544 | sparsity, 56                                    |
| 13504 | semantic parsing, 296                                 | 13545 | spectral learning, 129                          |
| 13505 | semantic role, 312                                    | 13546 | speech acts, 199                                |
| 13506 | Semantic role labeling, 312                           | 13547 | speech recognition, 137                         |
| 13507 | semantic role labeling, 418, 426                      | 13548 | split constituents, 318                         |
| 13508 | semantic underspecification, 307                      | 13549 | spurious ambiguity, 232, 274, 280, 302          |
| 13509 | semantics, 252, 289                                   | 13550 | squashing function, 147                         |
| 13510 | semi-supervised learning, 107, 117, 350               | 13551 | squashing functions, 65                         |
| 13511 | semiring algebra, 184                                 | 13552 | stand-off annotations, 101                      |
| 13512 | semiring notation, 209                                | 13553 | Stanford Natural Language Inference corpus, 356 |
| 13513 | semisupervised, 88                                    | 13554 | statistical learning theory, 42                 |
| 13514 | senses, 315   | 13555 | statistical machine translation, 438            |
| 13515 | sentence (logic), 294                                 | 13556 | statistical significance, 96                    |
| 13516 | sentence compression, 469                             | 13557 | stem, 18  |
| 13517 | sentence fusion, 470                                  | 13558 | stem (morphology), 205                          |
| 13518 | sentence summarization, 468                           | 13559 | stemmer, 90                                     |
| 13519 | sentiment, 81   | 13560 | step size, 53, 488                              |
| 13520 | sentiment lexicon, 32                                 | 13561 | stochastic gradient descent, 45, 54             |
| 13521 | sequence-to-sequence, 449                             | 13562 | stoplist, 92                                    |
| 13522 | shift-reduce parsing, 260                             | 13563 | stopwords, 92                                   |
| 13523 | shifted positive pointwise mutual information, 348    | 13564 | string (formal language theory), 201            |
| 13524 | shortest-path algorithm, 207                          | 13565 | string-to-tree translation, 448                 |
| 13525 | sigmoid, 63   | 13566 | strong compositionality criterion (RST), 399    |
| 13526 | simplex, 127  | 13567 | strongly equivalent grammars, 220               |
| 13527 | singular value decomposition, 73                      | 13568 | structure induction, 182                        |
| 13528 | singular value decomposition (SVD), 116               | 13569 | structured attention, 467                       |
| 13529 | singular vectors, 74                                  | 13570 | structured perceptron, 172                      |
| 13530 | skipgram word embeddings, 346                         | 13571 | structured prediction, 30                       |
| 13531 | slack variables, 48                                   | 13572 | structured support vector machine, 173          |
| 13532 | slot filling, 422                                     | 13573 | subgradient, 45, 56                             |

- 13576 subjectivity detection, 83  
 13577 subordinating conjunctions, 190  
 13578 subordinating discourse relations, 397  
 13579 sum-product algorithm, 176  
 13580 summarization, 137, 401  
 13581 supersenses, 350  
 13582 supervised machine learning, 32  
 13583 support vector machine, 47  
 13584 support vectors, 47  
 13585 surface form, 213  
 13586 surface realization, 461  
 13587 synchronous context-free grammar, 447  
 13588 synonymy, 86, 335  
 13589 synset (synonym set), 86  
 13590 synsets, 380  
 13591 syntactic dependencies, 264  
 13592 syntactic path, 319  
 13593 syntactic-semantic grammar, 297  
 13594 syntax, 187, 221, 289
- 13595 tagset, 188  
 13596 tanh activation function, 65  
 13597 target domain, 122  
 13598 target language, 437  
 13599 Targeted sentiment analysis, 83  
 13600 tense, 189  
 13601 terminal symbols (context-free grammars), 219  
 13602 test set, 39, 107  
 13604 test statistic, 96  
 13605 text classification, 29  
 13606 text mining, 17  
 13607 text planning, 461  
 13608 thematic roles, 313  
 13609 third axiom of probability, 478  
 13610 third-order dependency parsing, 270  
 13611 TimeML, 427  
 13612 tokenization, 88, 197  
 13613 tokens, 34  
 13614 topic models, 17  
 13615 topic segmentation, 387
- 13616 trace (syntax), 232  
 13617 training set, 32, 107  
 13618 transduction, 202  
 13619 transfer learning, 130  
 13620 transformer architecture, 452  
 13621 transition features, 160  
 13622 transition system, 275  
 13623 transition-based parsing, 235, 260  
 13624 transitive closure, 371  
 13625 translation error rate (TER), 441  
 13626 translation model, 138  
 13627 transliteration, 455  
 13628 tree-adjoining grammar, 230  
 13629 tree-to-string translation, 448  
 13630 tree-to-tree translation, 448  
 13631 treebank, 248  
 13632 trellis, 162, 215  
 13633 trigrams, 40  
 13634 trilexical dependencies, 257  
 13635 tropical semiring, 185, 209  
 13636 True negative, 93  
 13637 True positive, 93  
 13638 true positive, 481  
 13639 true positive rate, 95  
 13640 truncated singular value decomposition, 340  
 13641 truth conditions, 295  
 13643 tuning set, 39, 93  
 13644 Turing test, 15  
 13645 two-tailed test, 97  
 13646 type systems, 300  
 13647 type-raising, 231, 300  
 13648 types, 34
- 13649 unary closure, 239  
 13650 unary productions, 220  
 13651 underfit, 38  
 13652 underflow, 33  
 13653 undergeneration, 213, 223  
 13654 Universal Dependencies, 188, 263  
 13655 unlabeled precision, 240

- 13656 unlabeled recall, 240  
13657 unseen word, 181  
13658 unsupervised, 88  
13659 unsupervised learning, 83, 107  
13660 utterances, 199  
13661 validation function (semantic parsing),  
    306  
13663 validity (logic), 295  
13664 value function, 473  
13665 value iteration, 473  
13666 vanishing gradient, 65  
13667 vanishing gradients, 149  
13668 variable (AMR), 325  
13669 variable (logic), 293  
13670 variable, bound (logic), 294  
13671 variable, free (logic), 293  
13672 variance, 99  
13673 variance (learning theory), 38  
13674 variational autoencoder, 469  
13675 Vauquois Pyramid, 438  
13676 verb phrase, 223  
13677 VerbNet, 314  
13678 verbs, 189  
13679 vertical Markovization, 253  
13680 Viterbi algorithm, 160  
13681 Viterbi variable, 162  
13682 volition (semantics), 313  
13683 WARP loss, 417  
13684 weakly equivalent grammars, 220  
13685 weight decay, 71  
13686 weighted context-free grammar, 242  
13687 weighted context-free grammars, 228,  
    238  
13688 weighted finite state acceptors, 207  
13689 wide convolution, 77  
13691 Wikification, 411  
13692 Winograd schemas, 15  
13693 word embedding, 67  
13694 word embeddings, 61, 148, 336, 337  
13695 word representations, 336  
13696 word sense disambiguation, 85  
13697 word senses, 85  
13698 word tokens, 88  
13699 WordNet, 20  
13700 world model, 290  
13701 yield (context-free grammars), 219  
13702 zero-one loss, 44  
13703 Zipf's law, 155